Data Analysis Report: Life Expectancy Dataset

Author: Syed Bokhari

Date of Submission: 15th November 2022

Introduction:

The purpose of this project is to analyze the dataset Life Expectancy Dataset using Linear Regression. The dataset used for this project was acquired from Kaggle. The dataset was collected from WHO (World Health Organisation) tasked by the United Nations (UN) to look after matters related to health. The WHO keeps track of multiple health related factors and health status across multiple countries under the Global Health Observatory (GHO) data repository. It regularly makes this data publicly available for health-related research such as in this dataset. This dataset contains data on various health related factors for 193 countries over the period 2000-2015. The dataset and related details can be found at the link: https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who

The dataset has 22 columns and 2938 rows. There were several rows that had missing values for The columns and their details are listed below:

Year (int): Year where the data is from

Country (object): Name of the country the data is from

Status (object): Development status of the country. Can be Developing or Developed.

Life expectancy (float): Life expectancy in age (years)

Adult Mortality (float): Adult Mortality Rates of both sexes (probability of dying between 15 and 60 years per 1000 population)

infant deaths (int): Number of Infant Deaths per 1000 population

Alcohol (float): Alcohol, recorded per capita (15+) consumption (in litres of pure alcohol)

percentage expenditure (float): Expenditure on health as a percentage of Gross Domestic Product per capita (%)

Hepatitis B (float): Hepatitis B (HepB) immunization coverage among 1-year-olds (%)

Measles (int): Measles - number of reported cases per 1000 population

BMI (float): Average Body Mass Index of entire population

under-five deaths (int): Number of under-five deaths per 1000 population

Polio (float): Polio (Pol3) immunization coverage among 1-year-olds (%)

Total expenditure (float): General government expenditure on health as a percentage of total government expenditure (%)

Diphtheria (float): Diphtheria tetanus toxoid and pertussis (DTP3) immunization coverage among 1-year-olds (%)

HIV/AIDS (float): Deaths per 1 000 live births HIV/AIDS (0-4 years)

GDP (float): Gross Domestic Product per capita (in USD)

Population (float): Population of the country

thinness 1-19 years (float): Prevalence of thinness among children and adolescents for Age 10 to 19(%)

thinness 5-9 years (float): Prevalence of thinness among children for Age 5 to 9(%)

Income composition of resources (float): Human Development Index in terms of income composition of resources (index ranging from 0 to 1)

Schooling (float): Number of years of Schooling (years)

Objective:

The main objective of this analysis is to use the provided dataset to understand factors that impact life expectancy. The analysis involves making use of Linear Regression to look at the impact of these factors on Life Expectancy and determine through the results which factors have the most significant impact on Life Expectancy. Additional techniques for regression will be added to improve the ability of the models to explain the data. As such, the models' focus will be on interpretability of results instead of prediction. It is hoped that the insights gleaned from the results of the models will be able to determine which factors are the most important in explaining changes in Life Expectancy. The resulting correlations will be elaborated upon in the Key Insights section.

Data Cleaning:

The dataset was initially in csv format but was loaded into a Pandas dataframe. Pandas is a very useful library in Python for data preprocessing. It chiefly makes use of Dataframes, a built-in data structure, for performing data preprocessing and analysis in an efficient manner.

```
data = pd.read_csv('Life Expectancy Data.csv')
data.tail()
```

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	Diphtheria	HIV/AID
2933	Zimbabwe	2004	Developing	44.3	723.0	27	4.36	0.0	68.0	31	...	67.0	7.13	65.0	33.
2934	Zimbabwe	2003	Developing	44.5	715.0	26	4.06	0.0	7.0	998	...	7.0	6.52	68.0	36.
2935	Zimbabwe	2002	Developing	44.8	73.0	25	4.43	0.0	73.0	304	...	73.0	6.53	71.0	39.
2936	Zimbabwe	2001	Developing	45.3	686.0	25	1.72	0.0	76.0	529	...	76.0	6.16	75.0	42.
2937	Zimbabwe	2000	Developing	46.0	665.0	24	1.68	0.0	79.0	1483	...	78.0	7.10	78.0	43.

5 rows × 22 columns

Once the dataset had been loaded into a Pandas dataframe, a basic exploratory data analysis (EDA) was conducted. The EDA revealed one important fact. The data had lots of missing values.

```
data.shape
```

```
(2938, 22)
```

```
data.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  Income composition of resources  2771 non-null   float64
 21  Schooling                        2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB
```

Further inspection of the dataset revealed that several columns were missing many values.

```
data.isnull().sum()
```

```
Country                             0
Year                                0
Status                              0
Life expectancy                    10
Adult Mortality                    10
infant deaths                       0
Alcohol                           194
percentage expenditure              0
Hepatitis B                       553
Measles                             0
 BMI                               34
under-five deaths                   0
Polio                              19
Total expenditure                 226
Diphtheria                         19
 HIV/AIDS                           0
GDP                               448
Population                        652
 thinness  1-19 years              34
 thinness 5-9 years                34
Income composition of resources   167
Schooling                         163
dtype: int64
```

As missing data can be very problematic, a remedy was sought. Sadly, the columns which had too many missing values simply had to be dropped. There was no other way as interpolation of data would have skewed results and seeking data from alternative sources would have been a time-consuming process that may or may not have yielded results. Since the columns in question were not deemed critical to the analysis, their loss probably did not impact the project significantly. The columns 'Population', 'GDP' and 'Hepatitis B' were dropped.

```
data2 = data.drop(['Population','GDP', 'Hepatitis B'], axis=1)
```

```
data2.isnull().sum()
```

```
Country                             0
Year                                0
Status                              0
Life expectancy                    10
Adult Mortality                    10
infant deaths                       0
Alcohol                           194
percentage expenditure              0
Measles                             0
 BMI                               34
under-five deaths                   0
Polio                              19
Total expenditure                 226
Diphtheria                         19
 HIV/AIDS                           0
 thinness  1-19 years              34
 thinness 5-9 years                34
Income composition of resources   167
Schooling                         163
dtype: int64
```

The remaining columns still had significant missing values. An analysis of these columns was conducted which revealed that much of the missing data had to do with the latest year in the dataset, 2015. This was particularly true for columns 'Total expenditure' and 'Alcohol'.

```
missing2 = data[data['Total expenditure'].isnull()]
```

```
missing2['Year'].value_counts()
```

```
2015    181
2002      4
2001      4
2000      4
2011      3
2010      3
2009      3
2008      3
2007      3
2006      3
2005      3
2004      3
2003      3
2014      2
2013      2
2012      2
Name: Year, dtype: int64
```

```
missing2 = data[data['Alcohol'].isnull()]
```

```
missing2['Year'].value_counts()
```

```
2015    177
2005      2
2013      2
2014      1
2012      1
2011      1
2010      1
2009      1
2008      1
2007      1
2006      1
2004      1
2003      1
2002      1
2001      1
2000      1
Name: Year, dtype: int64
```

In other cases, the missing values seemed to correspond to specific countries which did not report relevant data. As a result, dropping rows with missing data was seen as the most viable option. As the data was for 16 years, dropping data for 1 year where many data values were missing was seen as the appropriate option. After the rows with missing values were dropped, the dataset was reduced from 2938 rows and 22 columns to 2556 rows and 19 columns.

```python
data3 = data2.dropna()
```

```python
data3.isnull().sum()
```

```
Country                            0
Year                               0
Status                             0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Measles                            0
 BMI                               0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
 HIV/AIDS                          0
 thinness  1-19 years              0
 thinness 5-9 years                0
Income composition of resources    0
Schooling                          0
dtype: int64
```

```python
data3.shape
```

```
(2556, 19)
```

The target variable was selected to be Life Expectancy. As such, descriptive columns such as 'Country' and 'Year' were not seen as particularly useful for the subsequent regression analysis which relied on numerical values. As such these columns were also dropped. The remaining data had 2556 rows and 17 columns. All its columns were numeric except Status which was a categorical column with 2 values only. There was an additional complication though.

```python
data3.drop('Year', axis=1,inplace=True)
```

```python
data3.drop('Country', axis=1, inplace=True)
```

```python
data3.shape
```

```
(2556, 17)
```

```python
data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2556 entries, 0 to 2937
Data columns (total 17 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Status                           2556 non-null   object 
 1   Life expectancy                  2556 non-null   float64
 2   Adult Mortality                  2556 non-null   float64
 3   infant deaths                    2556 non-null   int64  
 4   Alcohol                          2556 non-null   float64
 5   percentage expenditure           2556 non-null   float64
 6   Measles                          2556 non-null   int64  
 7    BMI                             2556 non-null   float64
 8   under-five deaths                2556 non-null   int64  
 9   Polio                            2556 non-null   float64
 10  Total expenditure                2556 non-null   float64
 11  Diphtheria                       2556 non-null   float64
 12   HIV/AIDS                        2556 non-null   float64
 13   thinness  1-19 years            2556 non-null   float64
 14   thinness 5-9 years              2556 non-null   float64
 15  Income composition of resources  2556 non-null   float64
 16  Schooling                        2556 non-null   float64
dtypes: float64(13), int64(3), object(1)
memory usage: 359.4+ KB
```

Many column names in the dataset had leading and trailing whitespaces. This would make querying columns for further analysis hectic. Consequently, these whitespaces from column names were removed.

```
colnames = data3.columns
colnames
```

```
Index(['Status', 'Life expectancy ', 'Adult Mortality', 'infant deaths',
       'Alcohol', 'percentage expenditure', 'Measles ', ' BMI ',
       'under-five deaths ', 'Polio', 'Total expenditure', 'Diphtheria ',
       ' HIV/AIDS', ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling'],
      dtype='object')
```

```
new_cols = []
for col in colnames:
    new_cols.append(col.strip())
new_cols
```

```
['Status',
 'Life expectancy',
 'Adult Mortality',
 'infant deaths',
 'Alcohol',
 'percentage expenditure',
 'Measles',
 'BMI',
 'under-five deaths',
 'Polio',
 'Total expenditure',
 'Diphtheria',
 'HIV/AIDS',
 'thinness  1-19 years',
 'thinness 5-9 years',
 'Income composition of resources',
 'Schooling']
```

```
rename_dict={}
for col, newcol in zip(colnames, new_cols):
    rename_dict[col] = newcol
rename_dict
```

```
{'Status': 'Status',
 'Life expectancy ': 'Life expectancy',
 'Adult Mortality': 'Adult Mortality',
 'infant deaths': 'infant deaths',
 'Alcohol': 'Alcohol',
 'percentage expenditure': 'percentage expenditure',
 'Measles ': 'Measles',
 ' BMI ': 'BMI',
 'under-five deaths ': 'under-five deaths',
 'Polio': 'Polio',
 'Total expenditure': 'Total expenditure',
 'Diphtheria ': 'Diphtheria',
 ' HIV/AIDS': 'HIV/AIDS',
 ' thinness  1-19 years': 'thinness  1-19 years',
 ' thinness 5-9 years': 'thinness 5-9 years',
 'Income composition of resources': 'Income composition of resources',
 'Schooling': 'Schooling'}
```

```
data3.rename(rename_dict, axis=1, inplace=True)
data3.columns
```

```
Index(['Status', 'Life expectancy', 'Adult Mortality', 'infant deaths',
       'Alcohol', 'percentage expenditure', 'Measles', 'BMI',
       'under-five deaths', 'Polio', 'Total expenditure', 'Diphtheria',
       'HIV/AIDS', 'thinness  1-19 years', 'thinness 5-9 years',
       'Income composition of resources', 'Schooling'],
      dtype='object')
```

The dataset was now prepared for the next step.

Feature Engineering:

As previously discussed, the Status column was the only remaining non-numeric column with two categorical values in it. As this column appeared useful for the analysis, it had to be one-hot encoded to be a part of the analysis as Regression models only accept numeric input. First, the data was divided into features and the target variable.

```
target = 'Life expectancy'
X = data3.drop(target, axis=1)
X.head()
```

	Country	Year	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	thinness 1-19 years	th
0	Afghanistan	2015	Developing	263.0	62	0.01	71.279624	1154	19.1	83	6.0	8.16	65.0	0.1	17.2	
1	Afghanistan	2014	Developing	271.0	64	0.01	73.523582	492	18.6	86	58.0	8.18	62.0	0.1	17.5	
2	Afghanistan	2013	Developing	268.0	66	0.01	73.219243	430	18.1	89	62.0	8.13	64.0	0.1	17.7	
3	Afghanistan	2012	Developing	272.0	69	0.01	78.184215	2787	17.6	93	67.0	8.52	67.0	0.1	17.9	
4	Afghanistan	2011	Developing	275.0	71	0.01	7.097109	3013	17.2	97	68.0	7.87	68.0	0.1	18.2	

```
y = data3[target]
y.head()
```

```
0    65.0
1    59.9
2    59.9
3    59.5
4    59.2
Name: Life expectancy, dtype: float64
```

Next, the categorical column was noted and passed through the ColumnTransformer object. This object had already been configured with the OneHotEncoder object. Both these objects as well as all subsequent Regression and Machine Learning objects and techniques are a part of the Sci-kit Learn library in Python.

```
#mask = (X.dtypes == np.object)
#categoricals = X.columns[mask]
categoricals = [key for key, value in X.dtypes.items()  if value=='O']
categoricals
```

```
['Status']
```

```
oec = ColumnTransformer(transformers=[("one_hot", OneHotEncoder(sparse=False), categoricals)], remainder='passthrough')
```

```
X.shape
```

```
(2556, 16)
```

```
oec.fit_transform(X[categoricals])
```

```
<2556x2 sparse matrix of type '<class 'numpy.float64'>'
	with 2556 stored elements in Compressed Sparse Row format>
```

Although this seemingly resolved the issue of the Status column it created an additional complication. The result from being passed through the Column Transformer is a Sparse Matrix, a data structure native to Sci-kit Learn. While efficient from a storage point of view, Sparse Matrices are not easily manipulated and for an interpretive project, they are not suitable. Consequently, the Pandas get_dummies function was used instead which yielded the desired results in the desired format.

```python
X = pd.get_dummies(data=X, columns=categoricals)
```

```python
X.shape
```

```
(2556, 17)
```

```python
X.head()
```

tage liture	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	thinness 1-19 years	thinness 5-9 years	Income composition of resources	Schooling	Status_Developed	Status_Developing
9624	1154	19.1	83	6.0	8.16	65.0	0.1	17.2	17.3	0.479	10.1	0	1
3582	492	18.6	86	58.0	8.18	62.0	0.1	17.5	17.5	0.476	10.0	0	1
9243	430	18.1	89	62.0	8.13	64.0	0.1	17.7	17.7	0.470	9.9	0	1
4215	2787	17.6	93	67.0	8.52	67.0	0.1	17.9	18.0	0.463	9.8	0	1
7109	3013	17.2	97	68.0	7.87	68.0	0.1	18.2	18.2	0.454	9.5	0	1

Regression Modeling:

Preliminaries

With the dataset in preprocessed form, the regression modeling could begin. First, the data was split into training and testing sets. The testing set was chosen to be 20% or 1/5th of the dataset. Throughout the rest of regression modeling analysis, these training and testing sets will be used.

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 17)
```

Next, a function for modeling contribution of features to regression results was defined. This will make explicit which features have the greatest contribution and thus provide easy interpretability of results.

Feature Selection Function

```python
[39]: def get_R2_features(model, features, type, test=True): 
    R_2_train=[]
    R_2_test=[]

    for feature in features:
        model.fit(X_train[[feature]],y_train)

        R_2_test.append(model.score(X_test[[feature]],y_test))
        R_2_train.append(model.score(X_train[[feature]],y_train))

    plt.bar(features,R_2_train,label="Train")
    plt.bar(features,R_2_test,label="Test")
    plt.xticks(rotation=90)
    plt.ylabel("$R^2$")
    plt.legend()
    plt.title(("Features with {} Regression").format(type))
    plt.show()
    print("Training R^2 mean value {} Testing R^2 mean value {} ".format(str(np.mean(R_2_train)),str(np.mean(R_2_test))) )
    print("Training R^2 max value {} Testing R^2 max value {} ".format(str(np.max(R_2_train)),str(np.max(R_2_test))) )
```

A list of all the feature names was also created.

```
features = X.columns
features
```

```
Index(['Adult Mortality', 'infant deaths', 'Alcohol', 'percentage expenditure',
       'Measles', 'BMI', 'under-five deaths', 'Polio', 'Total expenditure',
       'Diphtheria', 'HIV/AIDS', 'thinness  1-19 years', 'thinness 5-9 years',
       'Income composition of resources', 'Schooling', 'Status_Developed',
       'Status_Developing'],
      dtype='object')
```

Modeling with Linear Regression:

First, a Simple Linear Regression was run. A Pipeline object was defined with preprocessing and estimator steps. StandardScaler, a Sci-kit transformer object that 'normalizes' all data values by reducing them to a mean of 0 and with standard distribution of 1 was used for preprocessing. This will be a standard practice for pipelines in this project. For this instance, a simple Linear regression was used as an estimator.

Simple Linear Regression

```
steps = [('ss', StandardScaler()), ('lr', LinearRegression())]
```

```
pipe = Pipeline(steps)
```

```
pipe.fit(X_train, y_train)
```

```
Pipeline(memory=None,
     steps=[('ss', StandardScaler(copy=True, with_mean=True, with_std=True)), ('lr', LinearRegression(copy_X=True, fit_intercept=True, n_job
s=None,
         normalize=False))])
```

```
pipe.score(X_train, y_train)
```

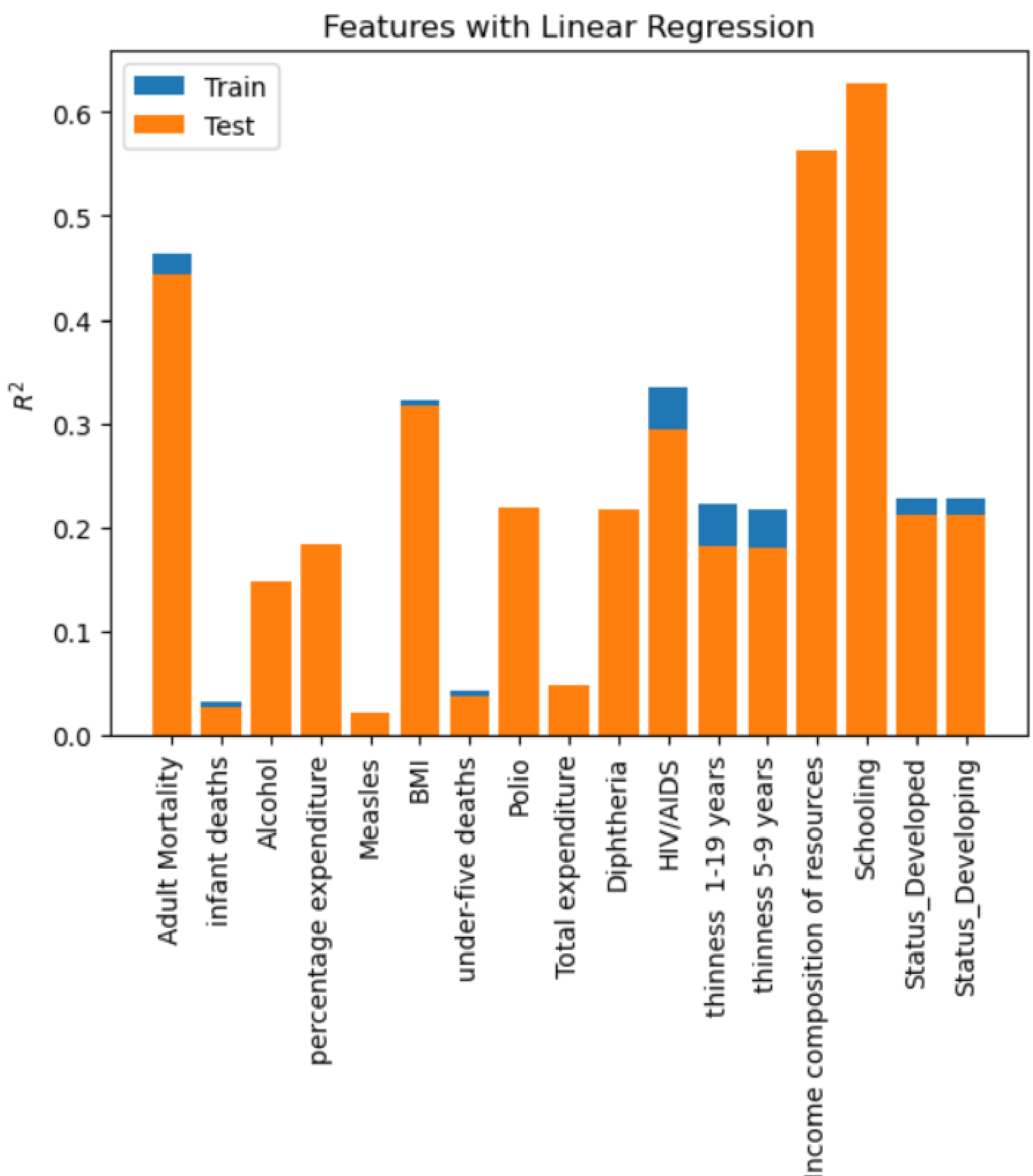
```
0.8331323206316779
```

```
pipe.score(X_test, y_test)
```

```
0.8268398257165515
```

The results indicated an R^2 score of 0.833 for the training set and 0.827 for the test set. This means that 83.3% of the variation in the training set and 82.7% of the variation in the test set could be explained by this model. Those are very good results.

Next, the contribution of features to the R^2 was noted.




```
Training R^2 mean value 0.2313135250147617 Testing R^2 mean value 0.23153307250743524
Training R^2 max value 0.5591224968322905 Testing R^2 max value 0.6272965574544296
```

This bar chart clearly displays the contribution of each feature to the R^2 score. 'Income composition of resources' and 'Adult Mortality' both feature highly here as 2nd and 3rd most important factor. Surprisingly, though, 'Schooling' is the most important feature in explaining life expectancy. This is an interesting development.

Modeling with Polynomial Regression:

Next, the regression model was made more complex by adding polynomial terms. The PolynomialFeatures object from Sci-kit learn was added to the pipeline object with degree of 2. The results are as follows:

Polynomial Regression

```
steps_poly = [('poly', PolynomialFeatures(degree=2)), ('ss', StandardScaler()), ('lr', LinearRegression())]
```

```
pipe_poly = Pipeline(steps_poly)
```

```
pipe_poly.fit(X_train, y_train)
```

```
Pipeline(memory=None,
     steps=[('poly', PolynomialFeatures(degree=2, include_bias=True, interaction_only=False)), ('ss', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('lr', LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False))])
```

```
pipe_poly.score(X_train, y_train)
```

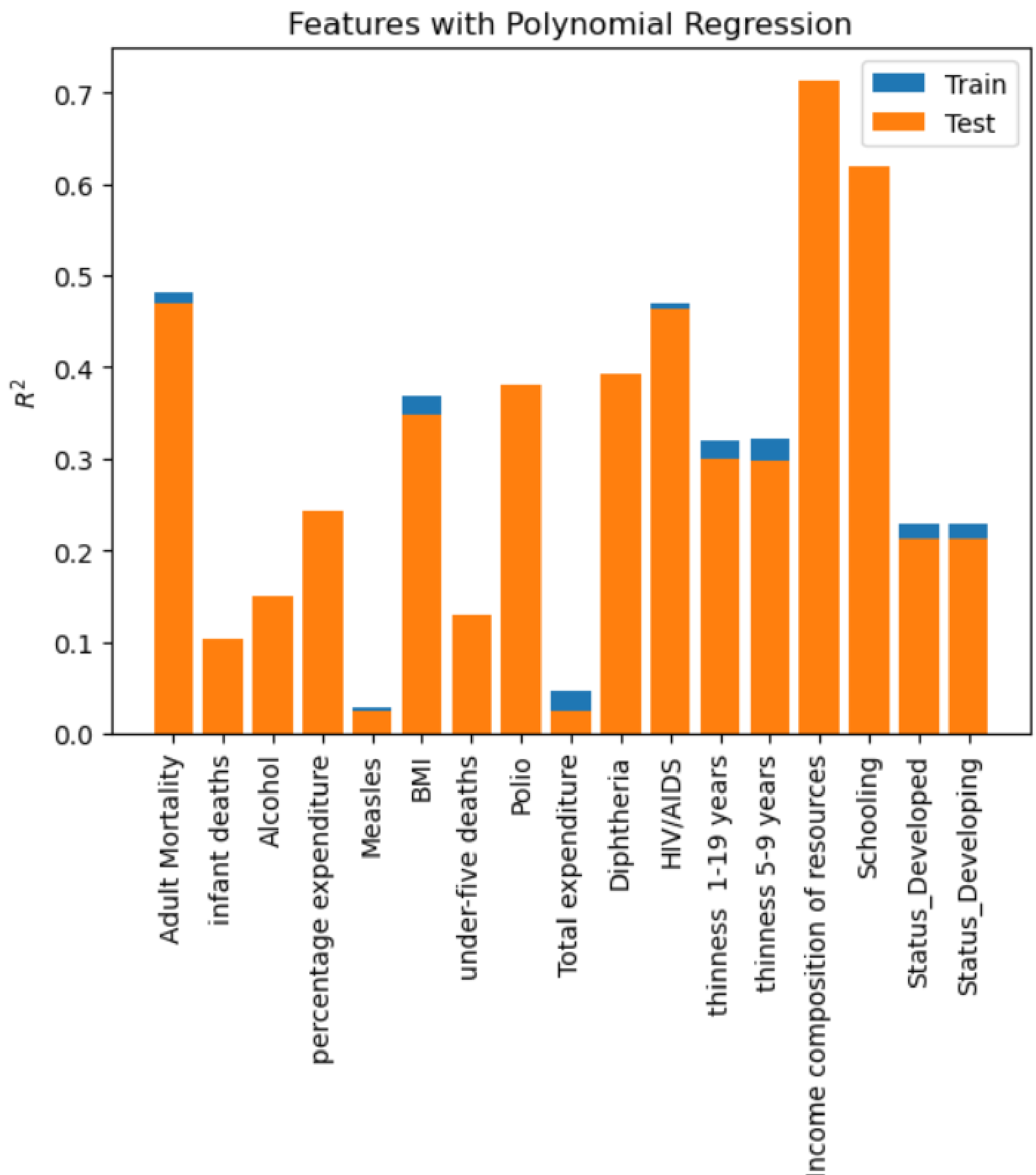
```
0.9346078362175404
```

```
pipe_poly.score(X_test, y_test)
```

```
0.8930712037468305
```

The results indicated an R^2 score of 0.935 for the training set and 0.893 for the test set. This means that 93.5% of the variation in the training set and 89.3% of the variation in the test set could be explained by this model. The addition of polynomial terms improved both training and testing scores, indicating that some of the variation in the data is due to non-linear effects.

Again, the contribution of features to the R^2 was observed.




```
Training R^2 mean value 0.29880237556927414 Testing R^2 mean value 0.29892043844523775
Training R^2 max value 0.7051788878479971 Testing R^2 max value 0.7123885545500187
```

Interestingly, the top 3 contributors to the R score are the same. However, now its 'Income composition of resources' that is the most important contributor and Schooling is the second most important contributor. However, Polynomial terms have increased the contribution of 'HIV/AIDS', 'Polio' and 'Diphtheria' to the results. It is possible they are capturing non-linear effects in these columns. But the model might simply be adding complexity where none is needed. Regularization can check for that.

Modeling with Ridge Regression:

In order to introduce some regularization in the model, Ridge Regression was used. To see the impact of Ridge Regression on the results, first a simple Ridge Regression was done without the Polynomial features.

Simple Ridge Regression

```
rr = Ridge(alpha=0.1)
```

```
rr.fit(X_train, y_train)
```

```
Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

```
rr.score(X_train, y_train)
```

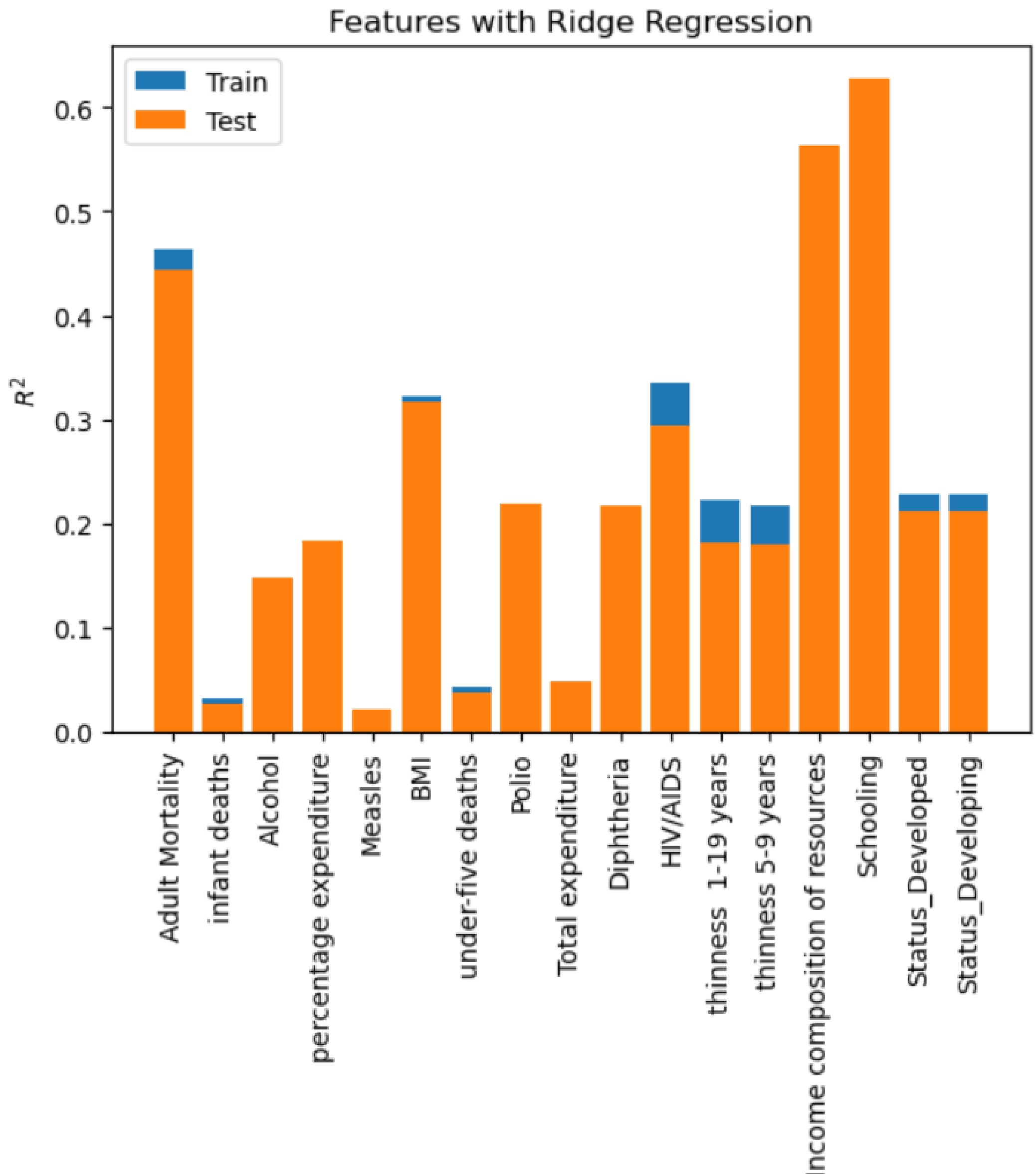
```
0.8331322594005357
```

```
rr.score(X_test, y_test)
```

```
0.8268248609747634
```

The results indicated an R^2 score of 0.833 for the training set and 0.827 for the test set. This means that 83.3% of the variation in the training set and 82.7% of the variation in the test set could be explained by this model.

Next, the contribution of features to the R^2 score was plotted.




```
Training R^2 mean value 0.23131348621193812 Testing R^2 mean value 0.23152430661853737
Training R^2 max value 0.5591224968198116 Testing R^2 max value 0.6272960306254108
```

The results are the same as in the case of the Simple Linear Regression model. Next, Ridge Regression must be performed with the addition of Polynomial features.

Modeling with Ridge Regression and Polynomial Features:

A pipeline feature was added and Ridge Regression was added as the estimator along with PolynomialFeatures in preprocessing steps. The results are:

Polynomial Ridge Regression

```
steps_ridge = [('poly', PolynomialFeatures(degree=2)), ('ss', StandardScaler()), ('rr', Ridge(alpha=0.1))]
```

```
pipe_ridge = Pipeline(steps_ridge)
```

```
pipe_ridge.fit(X_train, y_train)
```

```
Pipeline(memory=None,
     steps=[('poly', PolynomialFeatures(degree=2, include_bias=True, interaction_only=False)), ('ss', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('rr', Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001))])
```

```
pipe_ridge.score(X_train, y_train)
```

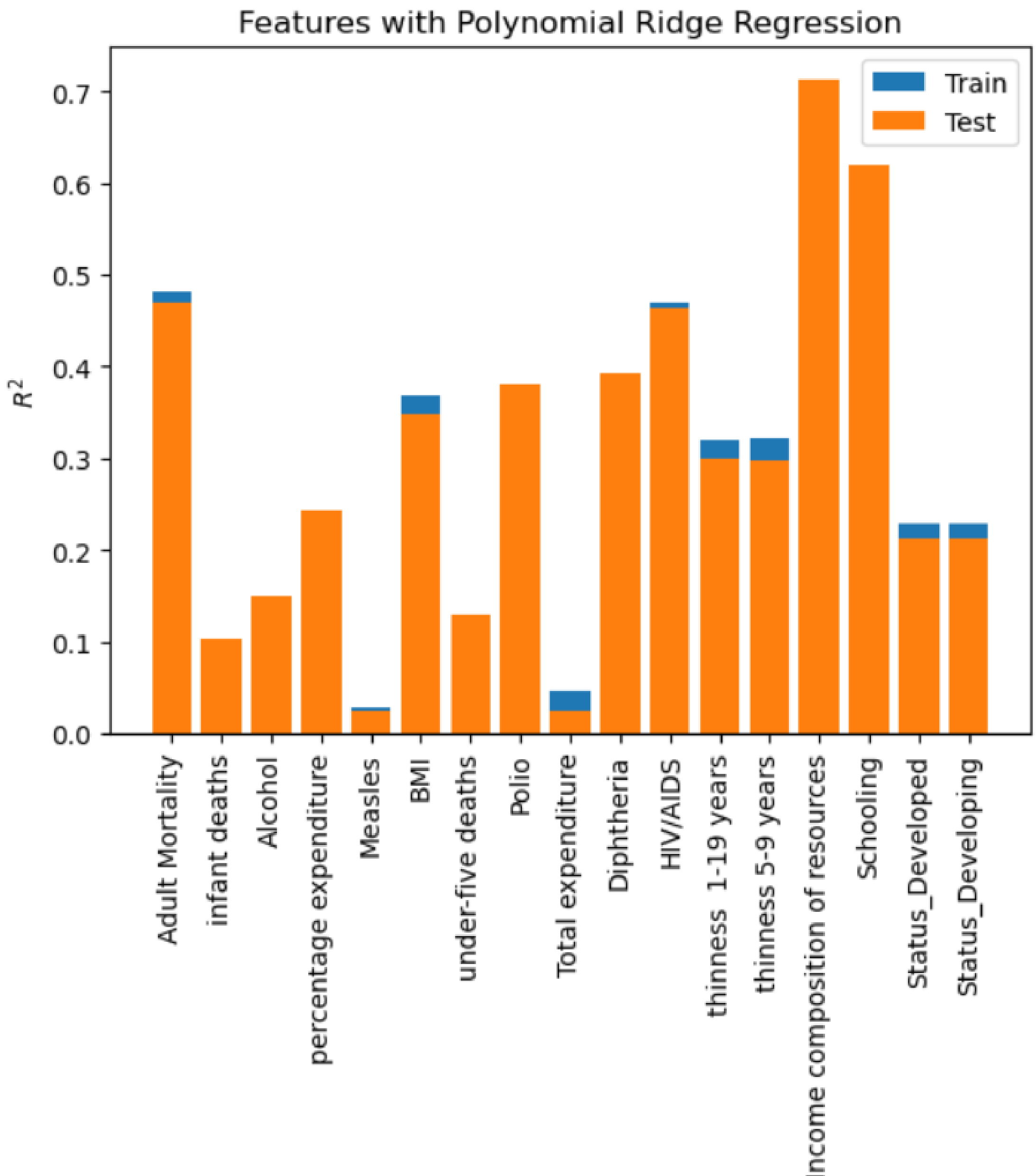
```
0.9343488885690111
```

```
pipe_ridge.score(X_test, y_test)
```

```
0.8996946553134787
```

The results indicated an R^2 score of 0.934 for the training set and 0.9 for the test set. This means that 93.4% of the variation in the training set and about 90% of the variation in the test set could be explained by this model. Again, the addition of polynomial terms improved both training and testing scores.

The contribution of features to the R^2 score was also plotted.




```
Training R^2 mean value 0.2988022992365552 Testing R^2 mean value 0.2989138541677465
Training R^2 max value 0.705178731879675 Testing R^2 max value 0.7124079257282122
```

These results are not very different from the case of simple Polynomial Regression. Next, Lasso regression may be used as an alternate regularization method.

Modeling with Lasso Regression:

Much like in the case of Ridge Regression, first an instance of simple Lasso Regression was used.

Simple Lasso Regression

```
las = Lasso(alpha=0.1)
```

```
las.fit(X_train, y_train)
```

```
Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

```
las.score(X_train, y_train)
```

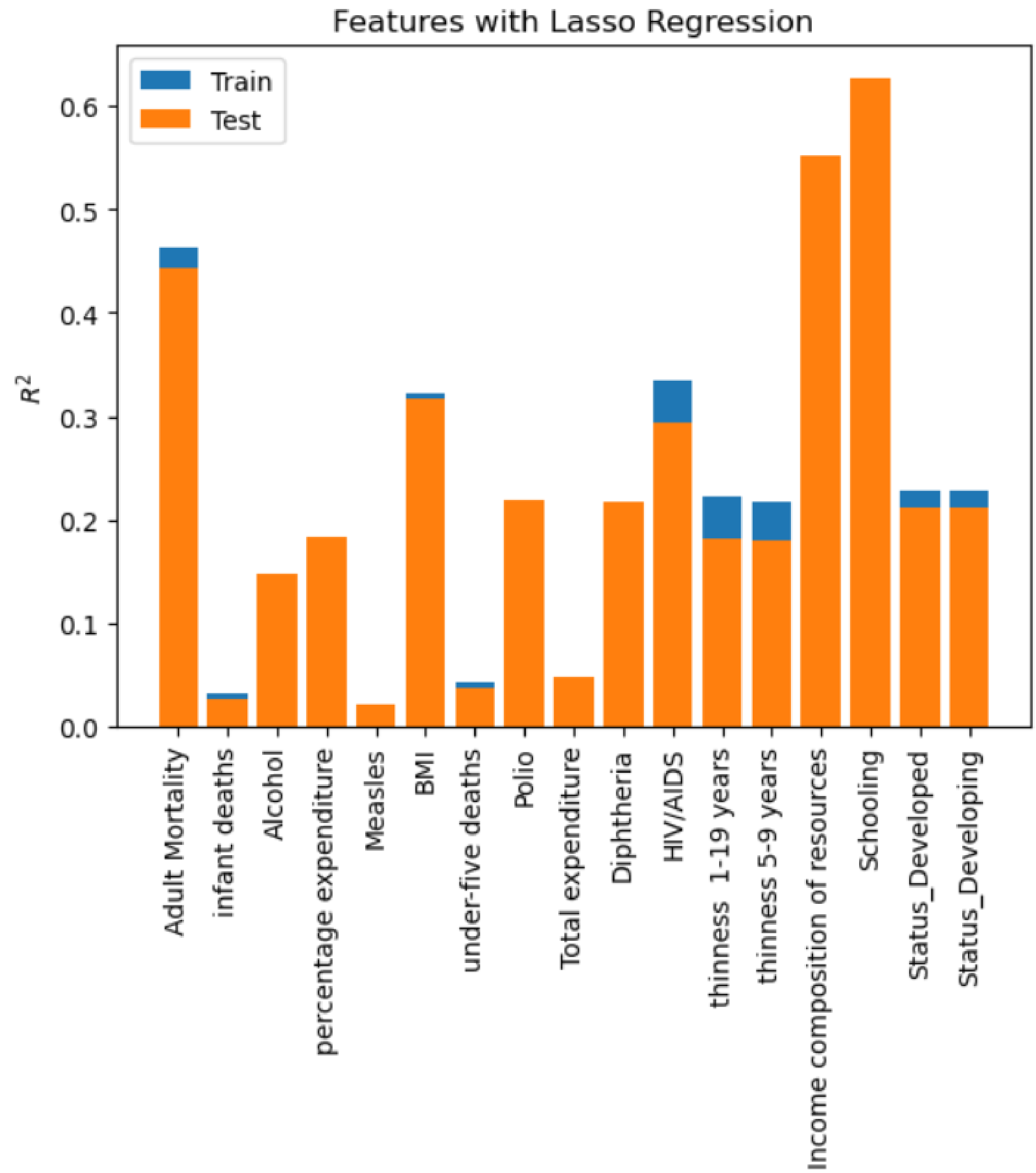
```
0.8256949819223256
```

```
las.score(X_test, y_test)
```

```
0.8166820647347979
```

The results indicated an R^2 score of 0.823 for the training set and 0.817 for the test set. This means that 82.3% of the variation in the training set and about 81.7% of the variation in the test set could be explained by this model. This is similar to simple Linear and simple Ridge regressions.

Feature contributions to the R^2 score were also noted.




```
Training R^2 mean value 0.23106271038129658 Testing R^2 mean value 0.23072593868258642
Training R^2 max value 0.5591114315165528 Testing R^2 max value 0.626790245478031
```

Here too, the results are similar to simple Linear and simple Ridge regressions. Next, lets add Polynomial effects to Lasso regression.

Modeling with Lasso Regression and Polynomial Features:

Polynomial features were added to Lasso Regression using Pipeline and the results were calculated.

Polynomial Lasso Regression

```
steps_lasso = [('poly', PolynomialFeatures(degree=2)), ('ss', StandardScaler()), ('las', Lasso(alpha=0.1))]
```

```
pipe_lasso = Pipeline(steps_lasso)
```

```
pipe_lasso.fit(X_train, y_train)
```

```
Pipeline(memory=None,
     steps=[('poly', PolynomialFeatures(degree=2, include_bias=True, interaction_only=False)), ('ss', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('las', Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False))])
```

```
pipe_lasso.score(X_train, y_train)
```

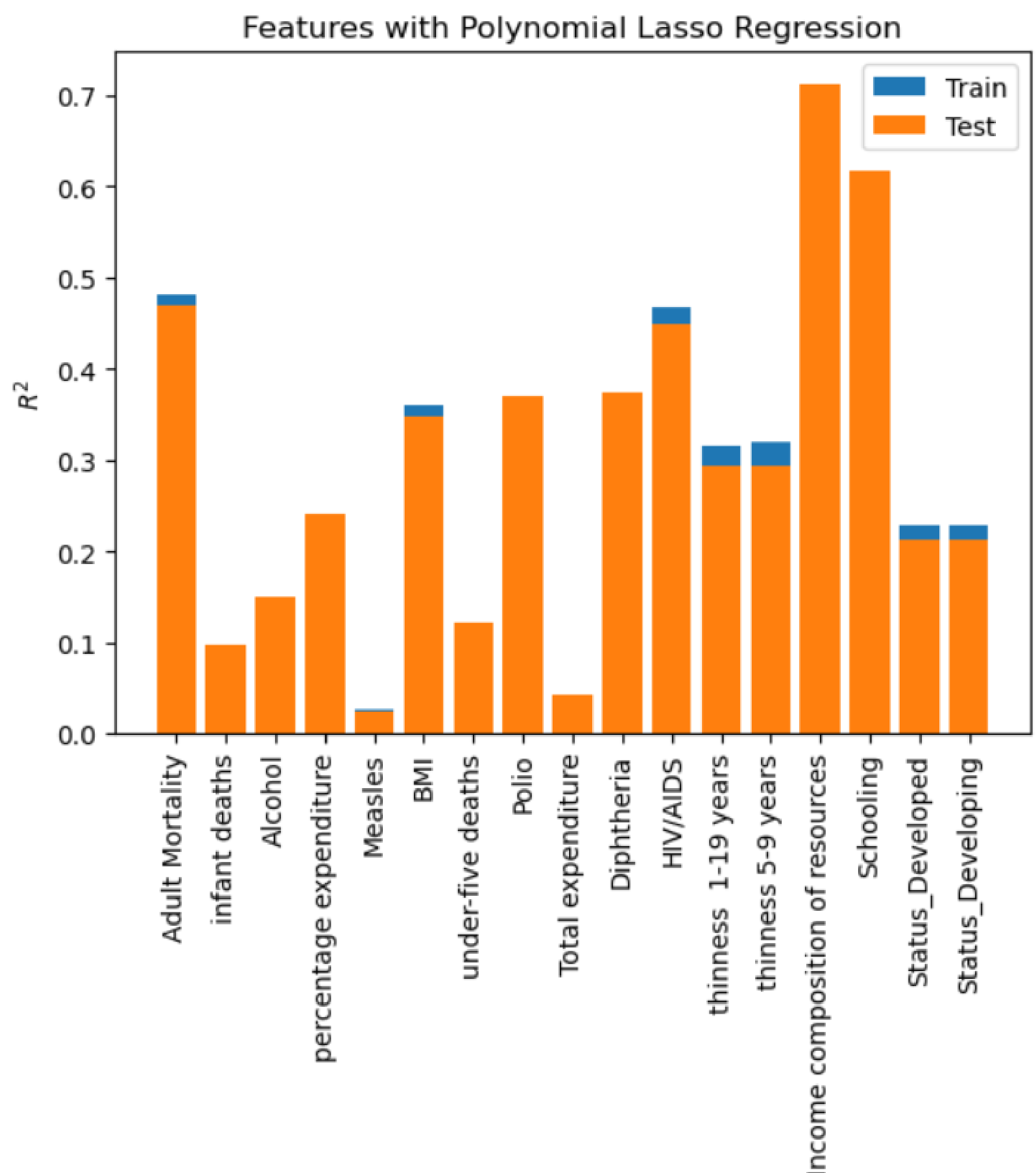
```
0.879495952969837
```

```
pipe_lasso.score(X_test, y_test)
```

```
0.8727481727165781
```

The results indicated an R^2 score of 0.88 for the training set and 0.873 for the test set. This means that 88% of the variation in the training set and about 87.3% of the variation in the test set could be explained by this model. These results are somewhat smaller than Ridge and Polynomial regressions. However, they are an improvement over the simple Lasso regression.

The features for this regression were also plotted:




```
Training R^2 mean value 0.29564428858135805 Testing R^2 mean value 0.29561252137409766
Training R^2 max value 0.6997476628321218 Testing R^2 max value 0.7119269098669276
```

These results are also similar to the Polynomial regression and Ridge with Polynomial Regression. Indeed, the contributions by the features are very close with the most important features being the same. Lasso may simply have reduced the contribution from smaller coefficients. Thus, all previous conclusions hold. It can be safely concluded that there are non-linear effects in the data that the model has successfully captured.

Ridge Regression with Optimal Alpha:

Previously, the Ridge Regression model used an alpha value of 0.1. Alpha is a hyperparameter, that can be tweaked to improve model performance. Alpha represents the degree of regularization with higher values representing greater regularization. The optimal alpha must provide enough regularization to ensure the model does not overfit, but not so much that important relationships are lost. To find the optimal alpha, a GridSearchCV object may be used. Provided with a list of possible values for hyperparameters, GridSearchCV iterates through all possibilities and finds the optimal result. This was attempted with alpha for Ridge Regression and degrees for PolynomialFeatures as hyperparameters with the following results:

Ridge Regression with Optimal Alpha

```
steps_ridge_cv = [('poly', PolynomialFeatures(degree=2)), ('ss', StandardScaler()), ('model', Ridge(alpha=0.1))]
```

```
pipe_ridge_cv = Pipeline(steps_ridge_cv)
```

```
param_grid = {
     "poly__degree": [1, 2, 3, 4, 5],
     "model__alpha": [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30]
}
```

```
search_ridge = GridSearchCV(pipe_ridge_cv, param_grid, n_jobs=2)
```

```
search_ridge.fit(X_train, y_train)
search_ridge
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('poly', PolynomialFeatures(degree=2, include_bias=True, interaction_only=False)), ('ss', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('model', Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001))]),
       fit_params=None, iid='warn', n_jobs=2,
       param_grid={'poly__degree': [1, 2, 3, 4, 5], 'model__alpha': [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
search_ridge.best_estimator_
```

```
Pipeline(memory=None,
     steps=[('poly', PolynomialFeatures(degree=2, include_bias=True, interaction_only=False)), ('ss', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('model', Ridge(alpha=3, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001))])
```

```
search_ridge.best_params_
```

```
{'model__alpha': 3, 'poly__degree': 2}
```

```
search_ridge.best_score_
```

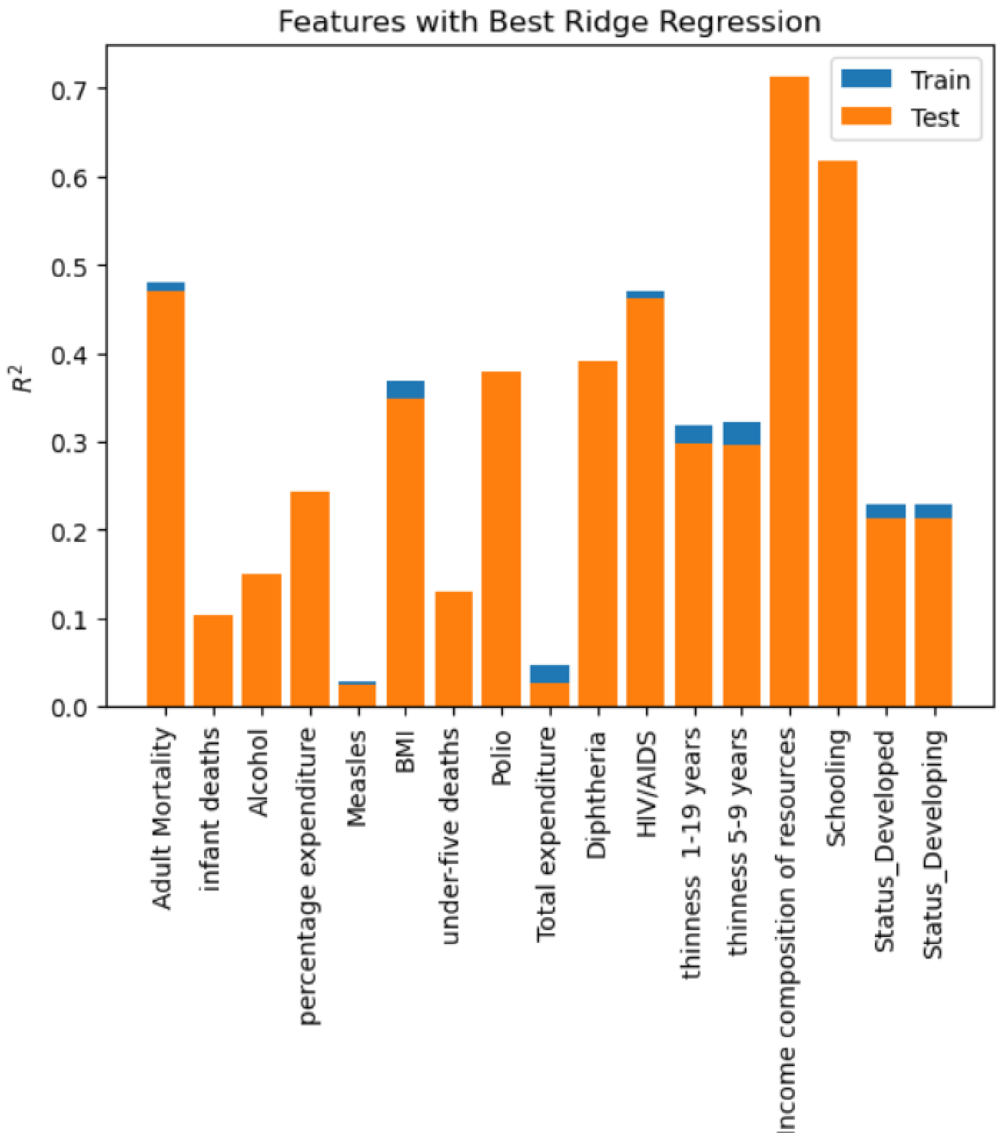
```
0.9130751959186046
```

```
best_ridge = search_ridge.best_estimator_
```

```
best_ridge.score(X_test, y_test)
```

```
0.9063399373699548
```

The results indicated an alpha value of 3 alongside polynomial values of degree 2 gave the best results. With these hyperparameters, the R^2 score was 0.913 for the training set and 0.906 for the test set. This means that 91.3% of the variation in the training set and about 90.6% of the variation in the test set could be explained by this model. These results are slightly better than the previous results with Polynomial Ridge Regression where alpha was 0.1.




```
Training R^2 mean value 0.29873917982698295 Testing R^2 mean value 0.29867185830453324
Training R^2 max value 0.7050474152681279 Testing R^2 max value 0.7128489905386017
```

These results are as expected. The most important features remain the same.

Lasso Regression with Optimal Alpha:

Similar to the case of Ridge Regression, Lasso Regression initially had an alpha value of 0.1 as well. An optimal value for alpha was sought for it as well. The results, with alpha and degrees for polynomial features as hyperparameters are given:

Lasso Regression with Optimal Alpha

```
steps_lasso_cv  = [('poly', PolynomialFeatures(degree=2)), ('ss', StandardScaler()), ('model', Lasso(alpha=0.1, tol=0.2, max_iter=100000))]
```

```
pipe_lasso_cv = Pipeline(steps_lasso_cv)
```

```
param_grid = {
     "poly__degree": [1, 2, 3, 4, 5],
     "model__alpha": [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30]
}
```

```
search_lasso = GridSearchCV(pipe_lasso_cv, param_grid, n_jobs=2)
```

```
search_lasso.fit(X_train, y_train)
search_lasso
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('poly', PolynomialFeatures(degree=2, include_bias=True, interaction_only=False)), ('ss', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('model', Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=100000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.2, warm_start=False))]),
       fit_params=None, iid='warn', n_jobs=2,
       param_grid={'poly__degree': [1, 2, 3, 4, 5], 'model__alpha': [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
search_lasso.best_estimator_
```

```
Pipeline(memory=None,
     steps=[('poly', PolynomialFeatures(degree=5, include_bias=True, interaction_only=False)), ('ss', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('model', Lasso(alpha=0.05, copy_X=True, fit_intercept=True, max_iter=100000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.2, warm_start=False))])
```

```
search_lasso.best_params_
```

```
{'model__alpha': 0.05, 'poly__degree': 5}
```

```
search_lasso.best_score_
```

```
0.875181971534635
```

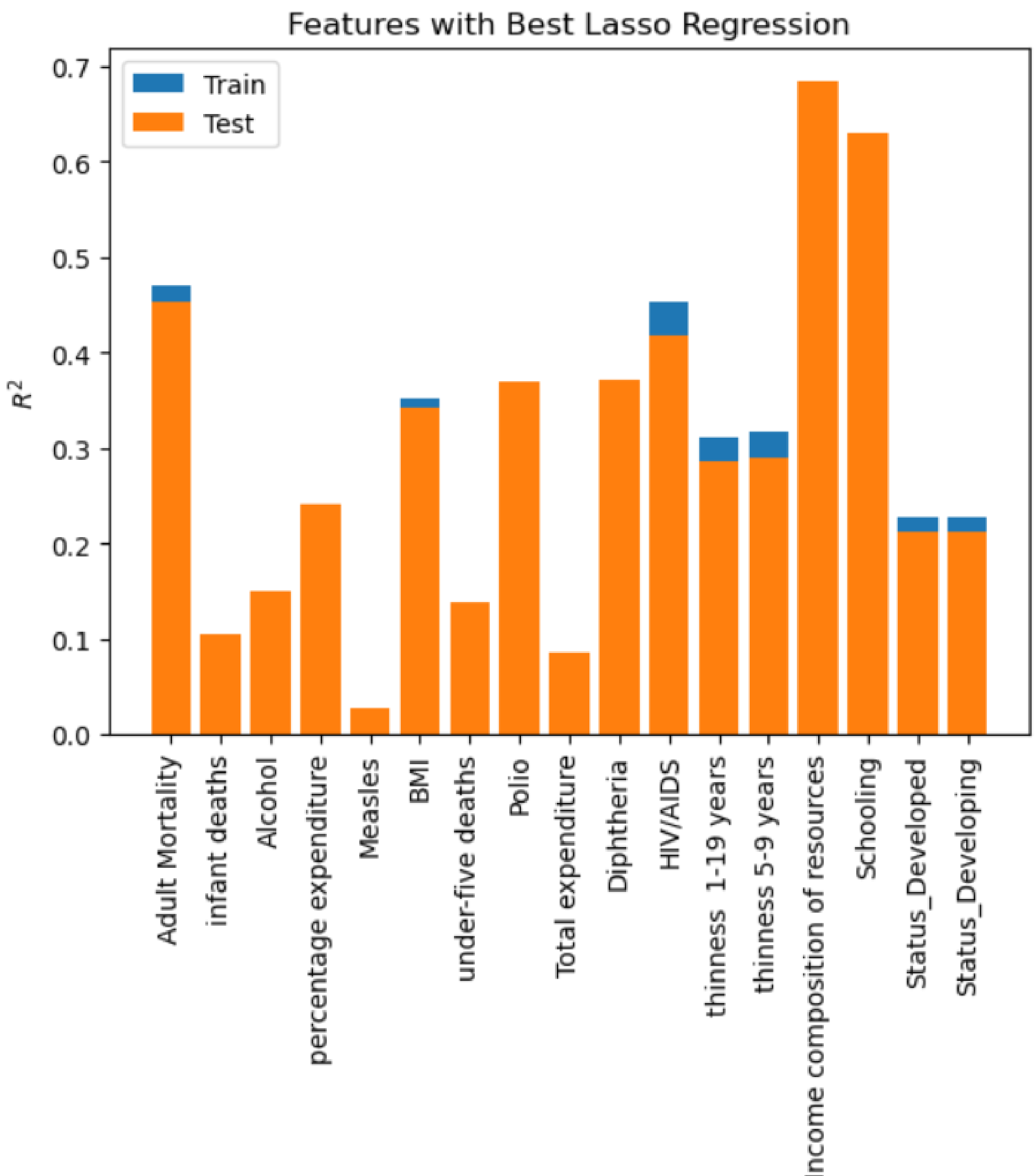
```
best_lasso = search_lasso.best_estimator_
```

```
best_lasso.score(X_test, y_test)
```

```
0.849849160816502
```

The results indicated an alpha value of 0.05 alongside polynomial values of degree 5 gave the best results. With these hyperparameters, the R^2 score was 0.875 for the training set and 0.85 for the test set. This means that 87.5% of the variation in the training set and about 85% of the variation in the test set could be explained by this model.

The feature contribution was plotted also:




```
Training R^2 mean value 0.2929376065076028 Testing R^2 mean value 0.29490375208343117
Training R^2 max value 0.6536293077196006 Testing R^2 max value 0.6839719029018225
```

These results are as expected. The most important features remain the same.

Elastic Net Regression with Optimal Alpha:

Ridge Regression uses L2 penalization while Lasso Regression uses L1 penalization. Elastic Net Regression is a version of regularization technique that mixes these two types of penalizations. In addition to alpha, it takes a hyperparameter, l1_ratio which determines the amount of L1 penalization. An optimal version of this regression was calculated with alpha, polynomial features' degrees and l1_ratio as hyperparameters. The results are as follows:

Elasticnet Regression

```
steps_elastic = [('poly', PolynomialFeatures(degree=2)), ('ss', StandardScaler()), ('model', ElasticNet(alpha=0.1, l1_ratio=0.1,tol=0.2, max
```

```
pipe_elastic = Pipeline(steps_elastic)
```

```
param_grid_elastic = {
    "poly__degree": [1, 2, 3, 4, 5],
    "model__alpha": [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30],
    "model__l1_ratio": [0, 0.1, 0.25, 0.5, 0.75, 0.9, 1]
}
```

```
search_elastic = GridSearchCV(pipe_elastic, param_grid_elastic, n_jobs=2)
```

```
search_elastic.fit(X_train, y_train)
search_elastic
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('poly', PolynomialFeatures(degree=2, include_bias=True, interaction_only=False)), ('ss', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('model', ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.1,
      max_iter=100000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.2, warm_start=False))]),
       fit_params=None, iid='warn', n_jobs=2,
       param_grid={'poly__degree': [1, 2, 3, 4, 5], 'model__alpha': [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30], 'model__l1_ratio': [0, 0.
1, 0.25, 0.5, 0.75, 0.9, 1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
search_elastic.best_estimator_
```

```
Pipeline(memory=None,
     steps=[('poly', PolynomialFeatures(degree=5, include_bias=True, interaction_only=False)), ('ss', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('model', ElasticNet(alpha=0.3, copy_X=True, fit_intercept=True, l1_ratio=0.1,
      max_iter=100000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.2, warm_start=False))])
```

```
search_elastic.best_params_
```

```
{'model__alpha': 0.3, 'model__l1_ratio': 0.1, 'poly__degree': 5}
```

```
search_elastic.best_score_
```

```
0.9071501689142346
```

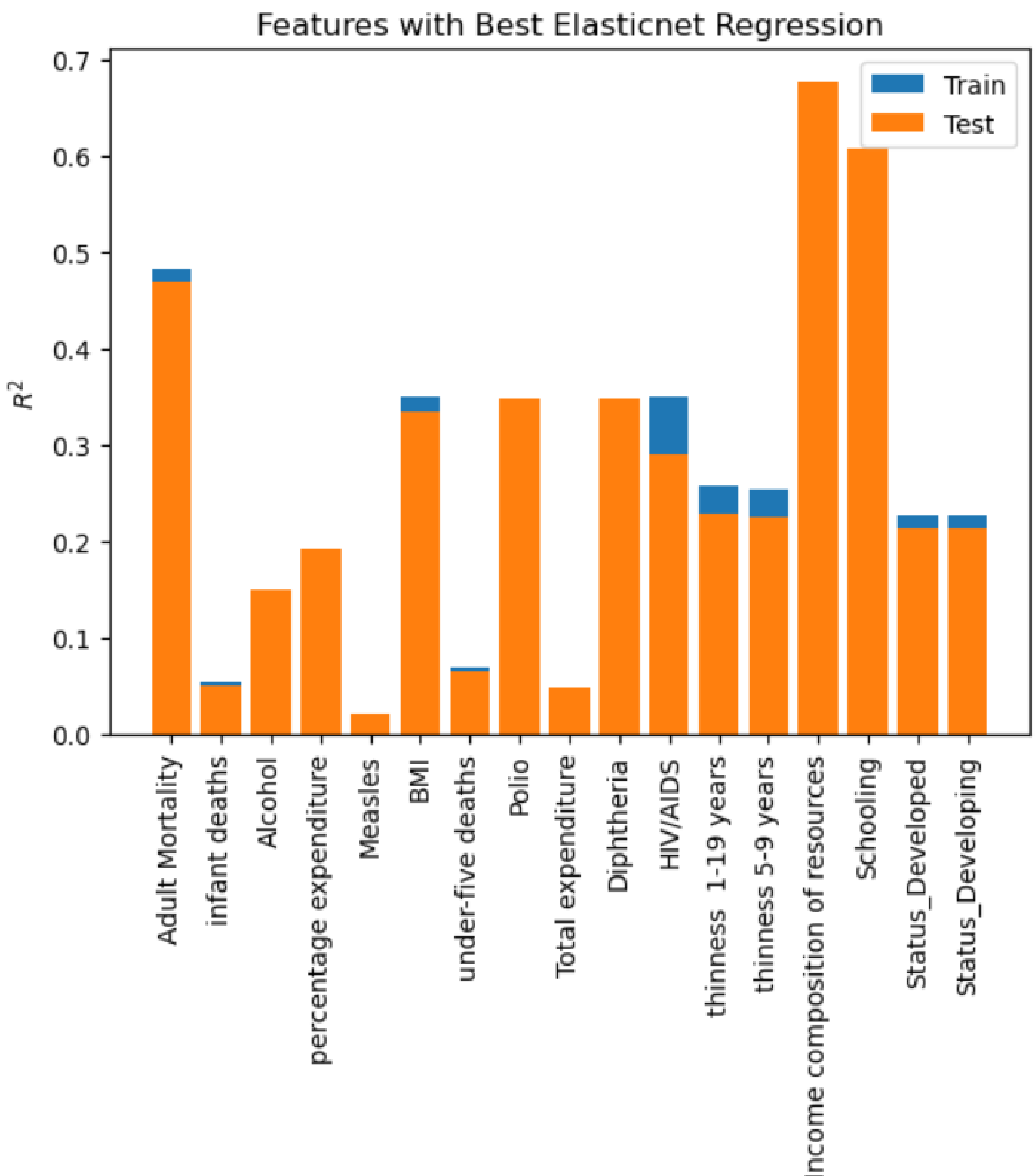
```
best_elastic = search_elastic.best_estimator_
```

```
best_elastic.score(X_test, y_test)
```

```
0.8515255237020576
```

The results indicated an alpha value of 0.3 alongside polynomial values of degree 5 and an l1_ratio of 0.1 gave the best results. With these hyperparameters, the R^2 score was 0.907 for the training set and 0.852 for the test set. This means that 90.7% of the variation in the training set and about 85.2% of the variation in the test set could be explained by this model. These results are good but slightly worse than ridge regression with best results. Furthermore, the model seemed to be overfitting to the training data as it performed worse on the test set relative to all the other models considered in the analysis.

The feature contributions were also plotted:




```
Training R^2 mean value 0.26719805001689767 Testing R^2 mean value 0.2633451751798665
Training R^2 max value 0.6567480883880727 Testing R^2 max value 0.6762706515913217
```

These results are not surprising as they match all earlier results. A small difference is reduction in 'HIV/AIDS' contribution for the test set relative to the training set.

Modeling Results:

The results from regression modeling indicated that all the models did a decent job of explaining the variations in the dataset. The worst was the simple Linear Regression model which is understandable as it added no complexity to the model. However, once

Polynomial features were added and regularization was performed, Ridge Regression gave the best results. Therefore, it may be considered the best model with respect to accuracy.

In terms of explainability, all the models did a decent job here also. As the top 3 features were never different for any model, it can be judged that all models were equally decent in this regard.

Key Findings and Insight:

The key finding from all these regressions were that:

1. Schooling is the one of the most important variables in explaining the difference in Life Expectancy. This may appear surprising at first glance. However, it seems to be a plausible response. As higher years of schooling may be correlated to greater education and awareness in the population leading to more informed health decisions. Higher schooling may also stand as a proxy for other desirable effects in society such as greater awareness of health-related issues, better healthcare provision etc.
2. Income composition of resources seems to be the most important variable in explaining Life Expectancy. This though important, is not surprising. As is evident, the greater the resources available, the higher the likelihood that a person will seek out medical care for any issues they face.
3. Adult Mortality Rate is the third most important variable in explaining Life Expectancy. This too is not surprising. It is evident that with less probability of dying for adults, the overall Life Expectancy would be higher.
4. Polio, HIV/AIDS, as well as Diphtheria seem to have some non-linear effects which can be captured through polynomial terms. These are important in explaining Life Expectancy. This is not particularly surprising as these diseases negatively impact Life Expectancy.
5. The most important insight from this analysis is the importance of education in improving Life Expectancy. Improved education outcomes can help countries with low Life Expectancy improve in addition to other key benefits.
6. A secondary insight is that minimizing the prevalence of diseases such as Polio, HIV/AIDS and diphtheria will also improve Life Expectancy.

Issues and Future Steps:

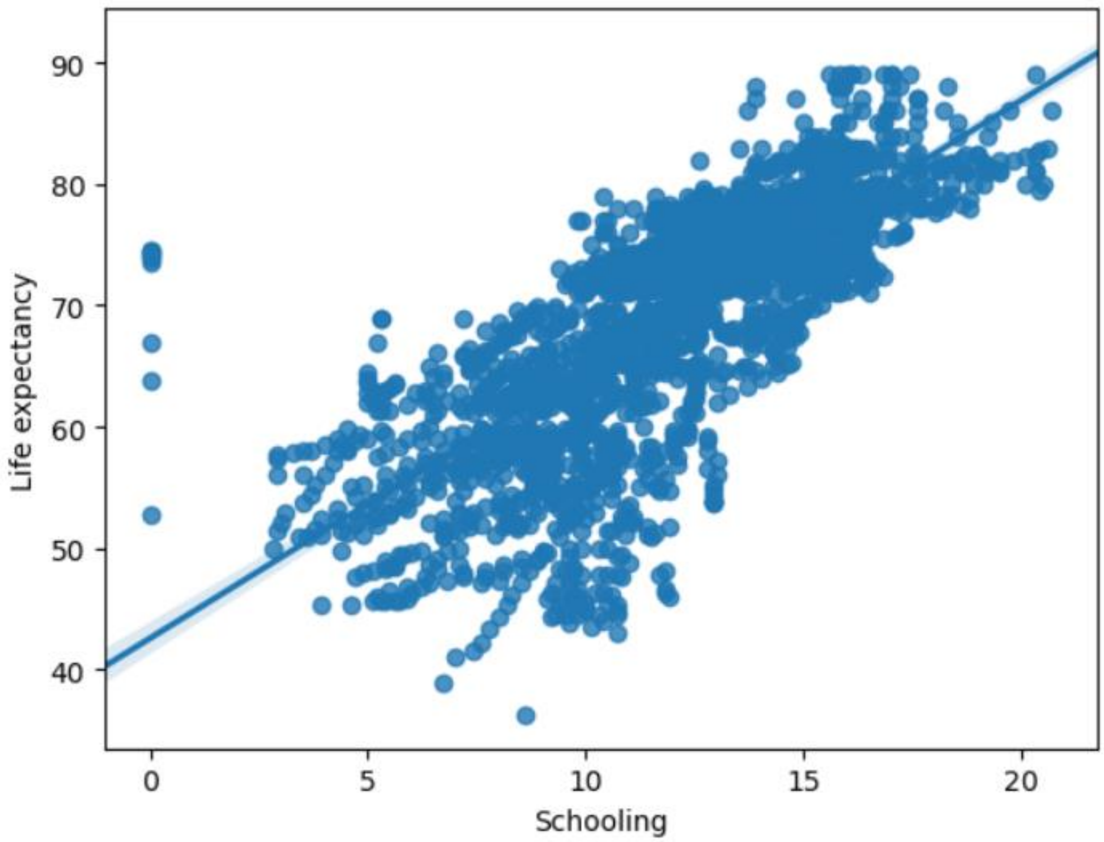
A key source of frustration while working with this dataset was the issue of missing values. This issue limited the analysis as the nature of the task at hand did not permit interpolation for missing values. This ultimately necessitated dropping entire columns and rows which was wasteful, albeit necessary. A better dataset would not have these missing values allowing for a more detailed analysis. A possible future step for this analysis would be to have data on additional factors impacting Life Expectancy, such as Prevalence of Non-Communicable Diseases (NCDs) for each country, and diet and nutrition related factors such as Prevalence of Malnutrition and Obesity. Such additional details could further explain the results of the model and factors influencing Life Expectancy, unpacking variables for further analysis. These could generate additional actionable insights. As it is, the most important insight from this analysis is the importance of education in improving Life Expectancy.

Bonus:

The plots of the three most important factors impacting Life Expectancy are given below:

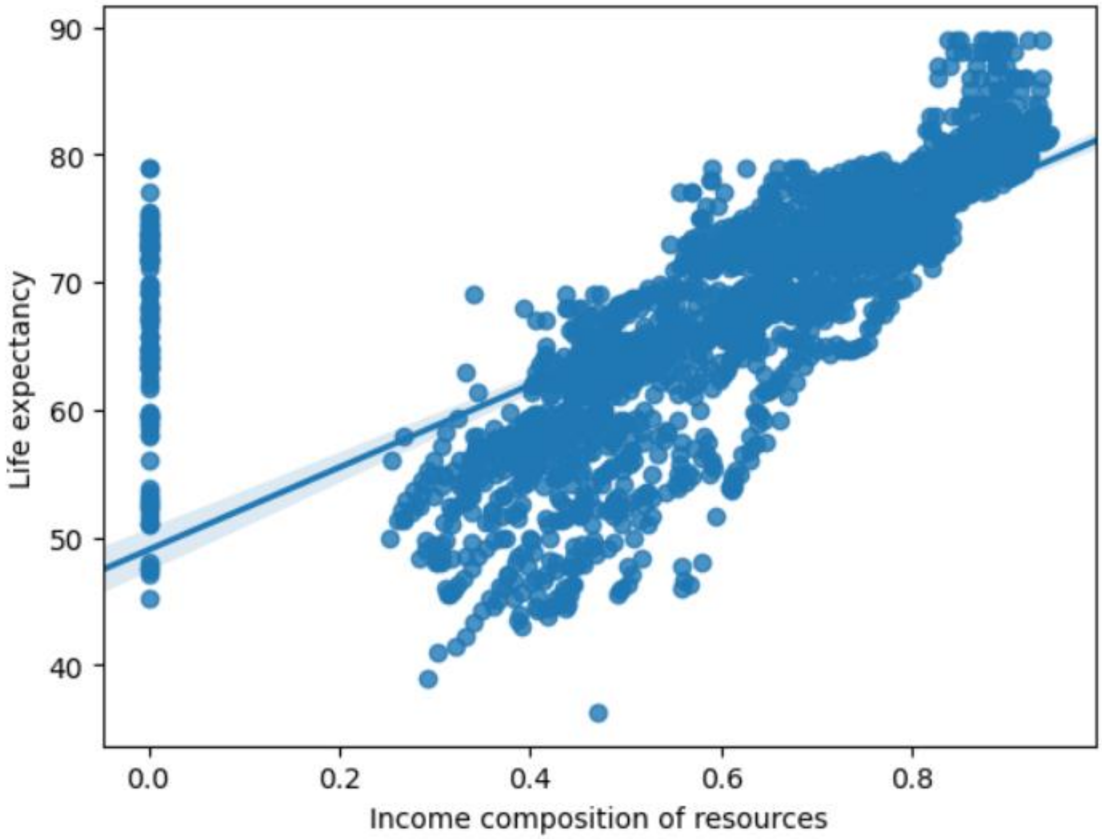
```
sns.regplot(x=X.Schooling, y=y)
```

```
<AxesSubplot:xlabel='Schooling', ylabel='Life expectancy'>
```



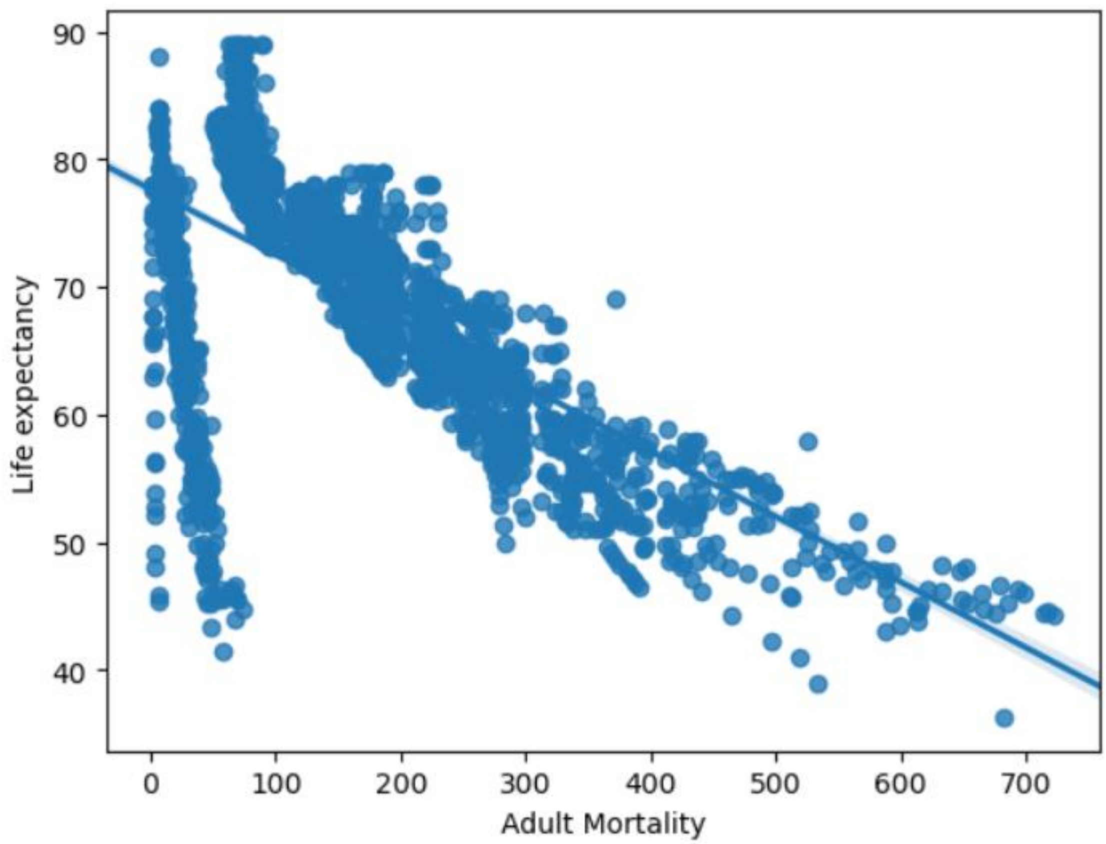
```
sns.regplot(x=X['Income composition of resources'], y=y)
```

```
<AxesSubplot:xlabel='Income composition of resources', ylabel='Life expectancy'>
```



```
sns.regplot(x=X['Adult Mortality'], y=y)
```

```
<AxesSubplot:xlabel='Adult Mortality', ylabel='Life expectancy'>
```



In addition to the various Regression models above, a few additional regressions were also attempted.

Theil-Sen Regression

```
from sklearn.linear_model import TheilSenRegressor
```

```
ts = TheilSenRegressor()
```

```
ts.fit(X_train, y_train)
```

```
TheilSenRegressor(max_subpopulation=10000)
```

```
ts.score(X_train, y_train)
```

```
0.5757228239722691
```

```
ts.score(X_test, y_test)
```

```
0.6726921917560384
```

Huber Regression

```
from sklearn.linear_model import HuberRegressor
```

```
hr = HuberRegressor(alpha=0.1)
```

```
hr.fit(X_train, y_train)
```

```
HuberRegressor(alpha=0.1)
```

```
hr.score(X_train, y_train)
```

```
0.04303974324108695
```

```
hr.score(X_test, y_test)
```

```
0.11244438159420378
```

RANSAC Regression

```
from sklearn.linear_model import RANSACRegressor
```

```
ran = RANSACRegressor()
```

```
ran.fit(X_train, y_train)
```

```
RANSACRegressor()
```

```
ran.score(X_train, y_train)
```

```
0.6740018202519784
```

```
ran.score(X_test, y_test)
```

```
0.6328657030866152
```

These regressions are based on more robust estimates of error and data corruption. They are meant to be less susceptible to outliers and error in the model. As can be seen, their results are not as good as the regression models considered in the analysis. But this may be because these regressions tend to divide data in inliers and outliers and deal with outliers differently (each regression shown here has its own way of dealing with them). As the data in the analysis has lots of data points with few outliers as shown in the plots at the start, these regressions may require certain preprocessing steps to be taken.