



UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING & INFORMATICS

CCI 501 Machine Learning

Title: MultiVariare Linear Regression - Life Expectancy

Submitted by		
	Member's Name	Registration No.
	Maureen Njoki	P61/41465/2021
	Dominic Motuka Monyoncho	P52/38272/2020
	Mugendi John Mwenda	P52/35253/2019

Introduction

The business question: Life Expectancy forecasting helps in understanding what factors are most responsible for short and long lifespans in countries.

The goal: Forecast life expectancy for a country.

How does this help UON ML 2022 Class?

- Gauge the technical skills required to prepare data and build a Linear regression ML model.
- Understand factors that influence life expectancy.

Assumptions

- The life expectancy can be described by the features in the dataset.

Using the data provided, Group F builds a life expectancy prediction model that is able to provide approximate expected life span based on the features provided.

A linear regression model is built to model life expectancy based on the features provided.

Root Mean Squared Error is used as an evaluation metric.

Data Preprocessing and Visualization

This project leverages the Life Expectancy data provided by the lecturer. Below is a brief description of the data.

```
data.info()
```

✓ 0.1s

Output exceeds the [size limit](#). Open the full output data [in a text](#)

```
<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
```

There is a mixture of object and numeric data types. There are missing values in some of the columns. A further look at the percentage distribution of the data points across the columns reveal that quite a number of columns have missing values.

```
The dataset has 22 columns.  
There are 14 columns that have missing values.
```

	Missing Values	% of Total Values
Population	652	22.2
Hepatitis B	553	18.8
GDP	448	15.2
Total expenditure	226	7.7
Alcohol	194	6.6
Income composition of resources	167	5.7
Schooling	163	5.5
BMI	34	1.2
thinness 1-19 years	34	1.2
thinness 5-9 years	34	1.2
Polio	19	0.6
Diphtheria	19	0.6
Life expectancy	10	0.3
Adult Mortality	10	0.3

A plot of life expectancy, which is the target variable, reveals the following:

1. This data is right tailed but seems to follow a normal distribution pattern.
2. The life expectancy steadily increases from late thirties to mid seventies.
3. Life expectancy between 70 and 75 seems the most frequent in the database.

Next step we check the correlation between life expectancy and all the other numeric variables.

Adult Mortality	-0.696359
HIV/AIDS	-0.556556
thinness 1-19 years	-0.477183
thinness 5-9 years	-0.471584
under-five deaths	-0.222529
infant deaths	-0.196557
Measles	-0.157586
Population	-0.021538
Year	0.170033
Total expenditure	0.218086
Hepatitis B	0.256762
percentage expenditure	0.381864
Alcohol	0.404877
GDP	0.461455
Polio	0.465556
Diphtheria	0.479495
BMI	0.567694
Income composition of resources	0.724776
Schooling	0.751975
Life expectancy	1.000000
Name: Life expectancy , dtype: float64	

The following was observed:

1. While we find Adult Mortality to be strongly negatively correlated with life expectancy, it seems to be an obvious reason because more adult deaths mean a lower life expectancy.
2. HIV/AIDs has the second strongest negative correlation with life expectancy while Schooling has the strongest positive correlation with life expectancy.
3. Interestingly, higher intake of alcohol, diphtheria, and BMI results in a higher life expectancy.
4. Population has the highest percentage of missing values, yet its influence on the target variable is insignificant.

Missing values in a dataset can have a significant impact on the performance of a machine-learning model. Depending on the amount and distribution of missing values, they can cause various problems, such as:

- **Reduced sample size:** If a large number of observations are missing, the sample size will be reduced, which can limit the amount of data available for training and evaluating the model.

- **Biased estimates:** If the missing values are not missing at random, the estimates of the model's parameters can be biased, leading to inaccurate predictions.
- **Reduced model performance:** Missing values can lead to reduced model performance, as the model may not be able to learn from all the available data.
- **Inconsistency:** Depending on the algorithm used, missing values can cause inconsistencies in the optimization process and lead to unstable or suboptimal solutions.

Several techniques can be used to handle missing values in a dataset, such as:

1. **Deleting observations:** Deleting observations with missing values can be a simple solution, but it can lead to a loss of information if many observations are removed.
2. **Imputing values:** Imputing missing values with a statistical measure such as the mean, median, or mode can be a simple solution, but it can also introduce bias and lead to inaccurate predictions.
3. **Using advanced imputation techniques:** Techniques such as multiple imputation or matrix completion can be used to impute missing values more sophisticatedly.
4. **Using models that are robust to missing values:** Some machine learning models, such as random forests or gradient boosting, can be robust to missing values and do not require imputation.

Deleting observations with missing values, also known as listwise deletion, is considered the best way to handle missing values in certain situations because it can lead to unbiased estimates and improved model performance. This is because when missing values are not missing at random, imputing values can lead to biased estimates of the model's parameters and inaccurate predictions. By removing observations with missing values, the model is only trained on complete cases, which can reduce the potential for bias.

However, listwise deletion can also have downsides. It can lead to a loss of information if a large number of observations are removed, which can reduce the sample size and limit the amount of data available for training and evaluating the model. Additionally, if the missing values are not missing completely at random, deleting observations can lead to biased estimates in the sample that is used to train and evaluate the model.

In this case however, we choose to delete the missing values because according to Pearson's correlation analysis, the variables with the highest missing values do not have a significant influence on the target variable.

```

# Discard columns with missing values more than 20%

missing_df = missing_values_table(data);
missing_columns = list(missing_df[missing_df['% of Total Values'] > 20].index)
print('\n', 'We will remove %d column(s).' % len(missing_columns))

# Drop the columns
data = data.drop(columns = list(missing_columns))

```

✓ 0.8s

We use a statistical test to verify whether life expectancy has a normal distribution or not. A p-value smaller than 0.05 means the null hypothesis is rejected. As such, a p-value of 0.05 or greater means that the distribution is normal.

Life expectancy, therefore, does follow a normal distribution.

A Shapiro-Wilk test is also used to confirm whether the data has been drawn from a normal distribution. The results are similar to observations made using D'Agostino's K² Test.

```

from scipy import stats

_, p = stats.normaltest(data['Life expectancy '])
print('Normal Test', format(p, '.3f'))
print(p <= 0.05)

# Check with Shapiro - Wilk test
from scipy.stats import shapiro

_, p = shapiro(data['Life expectancy '])
print('Shapiro Test', format(p, '.3f'))
print(p <= 0.05)

```

✓ 0.7s

```

Normal Test nan
False
Shapiro Test 1.000
False

```

We also look for countries with the highest and lowest life expectancy, respectively on average, across all years.

```
data.groupby(['Country'])['Life expectancy '].mean().sort_values(ascending=False).head(20)
```

✓ 0.6s

Country	
Japan	82.53750
Sweden	82.51875
Iceland	82.44375
Switzerland	82.33125
France	82.21875
Italy	82.18750
Spain	82.06875
Australia	81.81250
Norway	81.79375
Canada	81.68750
Austria	81.48125
Singapore	81.47500
New Zealand	81.33750
Israel	81.30000
Greece	81.21875
Germany	81.17500
Netherlands	81.13125
United Kingdom of Great Britain and Northern Ireland	80.79375
Luxembourg	80.78125
Finland	80.71250

Name: Life expectancy , dtype: float64

```
data.groupby(['Country'])['Life expectancy '].mean().dropna().sort_values(ascending=False).tail(20)
```

✓ 0.7s

Country	
Burkina Faso	55.64375
Burundi	55.53750
Guinea-Bissau	55.36875
Equatorial Guinea	55.31250
Mali	54.93750
Cameroon	54.01875
Zambia	53.90625
South Sudan	53.87500
Mozambique	53.39375
Somalia	53.31875
Nigeria	51.35625
Swaziland	51.32500
Zimbabwe	50.48750
Côte d'Ivoire	50.38750
Chad	50.38750
Malawi	49.89375
Angola	49.01875
Lesotho	48.78125
Central African Republic	48.51250
Sierra Leone	46.11250

Name: Life expectancy , dtype: float64

Finally, we check the highest and lowest life expectancy grouped by year, respectively.


```
data.groupby(['Year', 'Country'])['Life expectancy '].mean().dropna().sort_values(ascending=False)
```

✓ 0.9s

Year	Country	Life expectancy
2004	Italy	89.0
2014	Portugal	89.0
	Finland	89.0
2007	Sweden	89.0
	France	89.0
2014	Germany	89.0
2007	Spain	89.0
2014	Belgium	89.0
2010	New Zealand	89.0
2008	France	89.0
2009	Norway	89.0
2011	Austria	88.0
2015	Slovenia	88.0
2006	Sweden	88.0
2012	Austria	88.0
2004	Iceland	88.0
2010	Netherlands	88.0
2005	Italy	88.0
2011	Luxembourg	88.0
2014	Greece	88.0

Name: Life expectancy , dtype: float64

High life expectancy is consistent with high income countries across the years while low life expectancy is lowest in African countries.

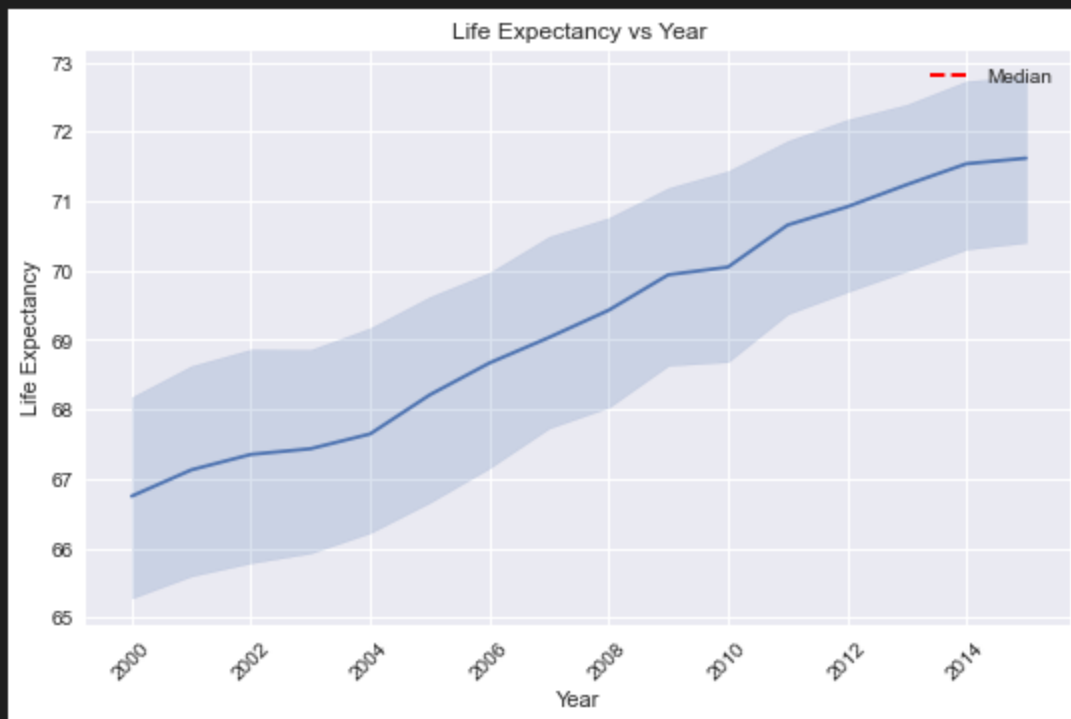
```
data.groupby(['Year', 'Country'])['Life expectancy '].mean().dropna().sort_
```

✓ 0.9s

Year	Country	
2004	Malawi	45.1
	Lesotho	44.8
2002	Zimbabwe	44.8
2001	Zambia	44.6
2003	Malawi	44.6
2005	Zimbabwe	44.6
	Lesotho	44.5
2003	Zimbabwe	44.5
2004	Zimbabwe	44.3
2006	Sierra Leone	44.3
2002	Malawi	44.0
2000	Zambia	43.8
2001	Malawi	43.5
2005	Sierra Leone	43.3
2000	Malawi	43.1
2004	Sierra Leone	42.3
2003	Sierra Leone	41.5
2001	Sierra Leone	41.0
2000	Sierra Leone	39.0
2010	Haiti	36.3

Name: Life expectancy , dtype: float64

Based on the Life Expectancy vs Year plot, there is a general increase in life expectancy across the years.



Multicollinearity

Collinearity, also known as multicollinearity, occurs when two or more predictor variables in a multiple regression model are highly correlated. This can cause problems in estimating and interpreting the model's parameters. When predictor variables are correlated, they explain some of the same variations in the response variable, and it becomes difficult to disentangle the individual effects of each predictor on the response.

When collinearity is present, the estimated regression coefficients can be imprecise, unstable, and difficult to interpret. The standard errors of the coefficients are typically inflated, which can lead to an increased risk of type I errors (false positives) in hypothesis testing.

The solution for multicollinearity can be dropping one or more correlated variables or using regularization methods like ridge or lasso regression, which can help reduce collinearity's impact by shrinking the coefficient estimates toward zero.

We drop values with a greater than or equal to 80% collinearity with the target variable.

```
# Remove the collinear features above a specified correlation coefficient
data = remove_collinear_features(data, 0.8);
```

✓ 0.1s

```
under-five deaths | infant deaths | 1.0
GDP | percentage expenditure | 0.9
```

Machine Learning

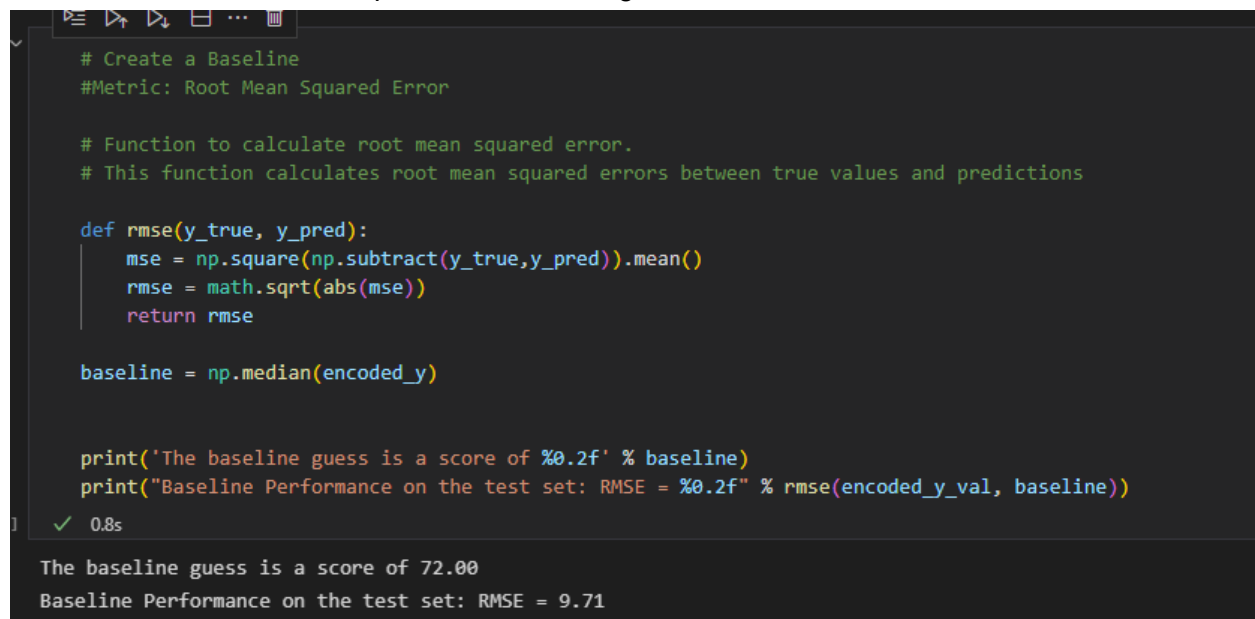
Cost Function

We use Root Mean Squared Error as the evaluation metric. It is a commonly used metric for evaluating the performance of a model when making continuous predictions.

It is the square root of the average squared differences between the predicted and actual values. It is recommended because it helps to penalize large errors more heavily than small errors, which is often desirable in practical applications.

Additionally, because RMSE is in the same unit as the target variable, it is easy to interpret and understand. Finally, it is differentiable and can be used to optimize the model.

We create a baseline score based on the median life expectancy value. We use this to evaluate how better our model is compared to a median guess.



```
# Create a Baseline
#Metric: Root Mean Squared Error

# Function to calculate root mean squared error.
# This function calculates root mean squared errors between true values and predictions

def rmse(y_true, y_pred):
    mse = np.square(np.subtract(y_true,y_pred)).mean()
    rmse = math.sqrt(abs(mse))
    return rmse

baseline = np.median(encoded_y)

print('The baseline guess is a score of %.2f' % baseline)
print("Baseline Performance on the test set: RMSE = %.2f" % rmse(encoded_y_val, baseline))
```

✓ 0.8s

The baseline guess is a score of 72.00
Baseline Performance on the test set: RMSE = 9.71

Since we only dropped data with missing values more than 20%, for the remaining values, we imputed based on the median. Median imputation is often preferred over other imputation methods, such as mean imputation because it is less sensitive to outliers.

Imputation

Mean imputation is sensitive to outliers, meaning that if there are extreme values in the data, the mean will be heavily influenced by these values. The imputed value may not be representative

of the majority of the data. In contrast, the median is a robust statistic, meaning that it is not affected by extreme values, and it is a better representation of the central tendency of the data.

Additionally, median imputation is preferred in situations where the distribution of the variable is skewed because the median is a better measure of central tendency for skewed distributions than the mean.

It's also important to note that median imputation does not change the distribution of the variable. It will not introduce bias in the model if the missing data are missing completely at random.

However, median imputation can also have downsides. It can lead to a loss of information if the missing values are not entirely missing at random, and it does not account for the relationship between the variable with missing values and the other variables in the dataset. More advanced imputation techniques, such as multiple imputations or using predictive models for imputation, can be more appropriate.

```
#Imputer object
imputer = Imputer(strategy = 'median')

#Fitting on train da
imputer.fit(encoded_X)

#Transform train and test data
encoded_X = imputer.transform(encoded_X)
encoded_X_val = imputer.transform(encoded_X_val)
```

✓ 0.7s

```
print('\n', 'Checking all data is finite')
print('-' * 30)
|
#Making sure all values are finite
print (np.where(~np.isfinite(encoded_X)))
print (np.where(~np.isfinite(encoded_X_val)))
```

✓ 0.6s

Data Scaling

We used normalization for scaling.

```
#Training, testing and evaluating a model
def train_test_evaluate(ml_model):
    print(ml_model)
    start = time.time()

    #Train

    # # Normalizing the features so that different units do not affect the algorithms.
    # While this process is not necessary for tree-based models, nevertheless, it is a good practice
    model = TransformedTargetRegressor(regressor= ml_model,
                                       transformer = MinMaxScaler()
                                       ).fit(encoded_X,encoded_y)

    #Test
    # Inverse transformation happens at this step
    model_pred = model.predict(encoded_X_val)

    #Evaluate
    model_rmse = rmse(encoded_y_val, model_pred)

    end = time.time()
    time_taken = end - start

    print('Time taken: %0.2f' %time_taken, 's.')

    #Return performance metric
    return model_rmse
```

✓ 0.8s

Normalization is a technique that scales the data to a fixed range, usually between 0 and 1. It is typically used when the data has a Gaussian (or normal) distribution or when the data's scale is unknown. Normalizing the data can make it easier to compare data from different sources or to compare data that has been measured in different units.

Normalization is preferred over standardization when the data has a skewed distribution because standardization can amplify the effect of outliers in the data. In contrast, normalization can make the data more robust to outliers. Additionally, normalization makes it easy to interpret the data as the values will be between 0 and 1.

Modeling

We run the linear regression model using its default parameters. Our RMSE was significantly lower than the baseline RMSE, which means we can use ML for life expectancy forecasting.

```
#Linear Regression
lr = LinearRegression()
lr_rmse = train_test_evaluate(lr)
print ('Linear Regression Root Mean Squared Error: %0.4f' %lr_rmse)
```

✓ 0.8s

LinearRegression()

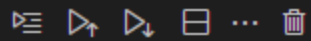
Time taken: 0.01 s.

Linear Regression Root Mean Squared Error: 4.1269

We also attempted to implement a linear regression model from scratch, including implementing the gradient descent function. The final RMSE was slightly higher than the RMSE with Scikit Learn.

Multivariate Linear Regression from Scratch

An attempt to implement the cost function and gradient descent from scratch



```
ones = np.ones([encoded_X.shape[0], 1])
print(ones)
encoded_X = np.concatenate((ones, encoded_X), axis=1)
encoded_y = encoded_y.values
```

✓ 0.8s

```
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
```

```
theta = np.zeros([1, encoded_X.shape[1]])
print(theta)

#Hyperparameters
alpha = 0.001 #Learning rate
iters = 10 #Number of iterations to perform gradient descent (epochs)
```

✓ 0.7s

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
# Cost function
def compute_cost(X,y,theta):
    inner = np.power((X @ theta.T)-y),2)
    return np.sum(inner)/(2 * len(X))
```

✓ 0.7s


```
# Gradient descent and cost function
gradient, cost = gradient_descent(encoded_X, encoded_y, theta, iter
print(gradient)
```

```
final_cost = compute_cost(encoded_X, encoded_y, gradient)
print(final_cost)
```

✓ 0.1s

1277228993317.8599

1.8791735078274533e+22

2.7823991507517284e+32

4.1197971971377284e+42

6.100033904634616e+52

9.032098391684809e+62

1.3373499661222575e+73

1.9801654657944706e+83

2.9319590019465247e+93

4.341245081580223e+103

[[-1.61330922e+43 -2.97443739e+45 -5.12249072e+45 -5.83730838e+43

-3.38959740e+45 -1.19860631e+45 -8.44677348e+47 -3.44630984e+44

-1.11917889e+45 -7.61082700e+43 -1.08691527e+45 -4.14891812e+43

-1.53143023e+44 -1.54800625e+44 -8.38148100e+42 -1.62932327e+44]]

4.341245081580223e+103

Conclusion

We find that machine learning can be applied to the process of life expectancy prediction.

Future steps would include tweaking the model parameters to see if its possible to improve the prediction capabilities.