Internship Title:-

Predicting Life Expectancy Using Machine Learning

Student name: Neeraj Patil

Email id :

1. neerajpatil14@gmail.com
2. 2017.neeraj.patil@ves.ac.in

Github link: https://github.com/SmartPracticeschool/llSPS-INT-1513-Predicting-Life-Expectancy-using-Machine-Learning

Index

1. INTRODUCTION

i. Overview

Life expectancy is one of the most important factors in end-of-life decision making. Good prognostication, for example, helps to determine the course of treatment and helps to anticipate the procurement of health care services and facilities, or more broadly: facilitates Advance Care Planning.

ii. Purpose

The goal of this project is to use techniques of data science to estimate life expectancies. To well equip hospitals with advanced health care units. The government would also take an important decision in the sectors and economy affected by this vital concept.

2. LITERATURE SURVEY

i. Existing problem

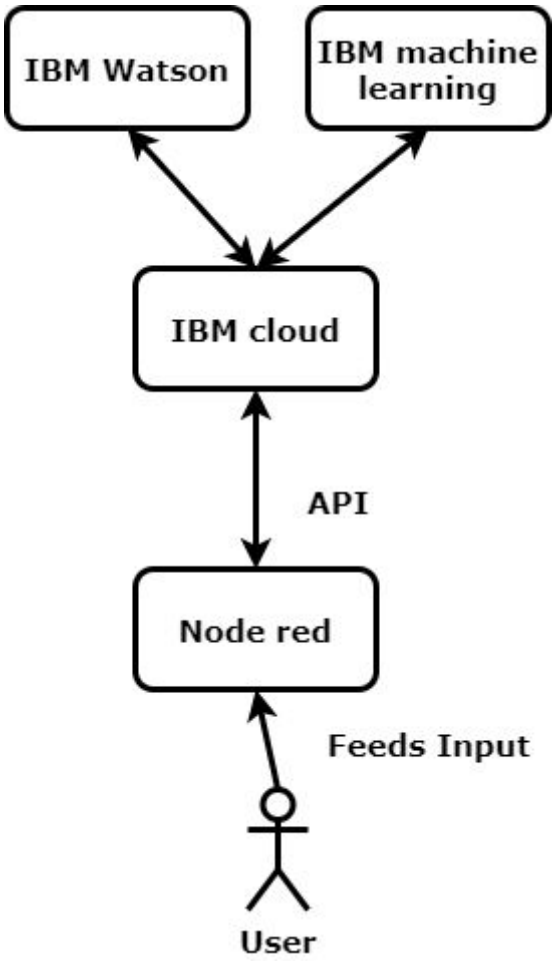
Prediction of individual life span represents a significant challenge for ageing research that is important for understanding factors influencing longevity, as well as identifying life-span–associated characteristics that can be studied as surrogates of longevity in laboratory experiments. Inadequate healthcare infrastructure to treat patients.

ii. Proposed solution

The solution consists of training the model with an accurate algorithm. Also deploying it on a user-friendly interface like the website. It can be applied for all living beings, different countries as a whole.
A central point of uncertainty is whether the kind of progress and development in many countries that has marked the past couple of centuries will continue until 2050.
Perhaps the future will be characterized by poverty, misery, and a shorter life expectancy. As people live longer, ages at death are becoming more similar.
This dual advance over the last two centuries, a central aim of public health policies, is a major achievement of modern civilization.
Some recent exceptions to the joint rise of life expectancy and life span equality, however, make it difficult to determine the underlying causes of this relationship.

3. THEORETICAL ANALYSIS

i. Block diagram



ii. Hardware / Software designing

The software is built using IBM cloud services and node-red dashboard.

Tools used for making this software

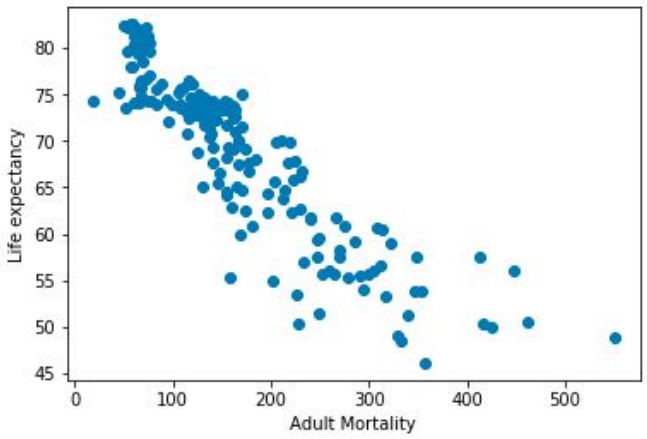
- IBM cloud
- IBM cloud services
- IBM Watson
- IBM machine learning
- Node-red
- Python
- Anaconda for notebook

4. EXPERIMENTAL INVESTIGATIONS

The dataset was important in analysis, calculations were performed using python Plotting life expectancy against adult mortality to find out the

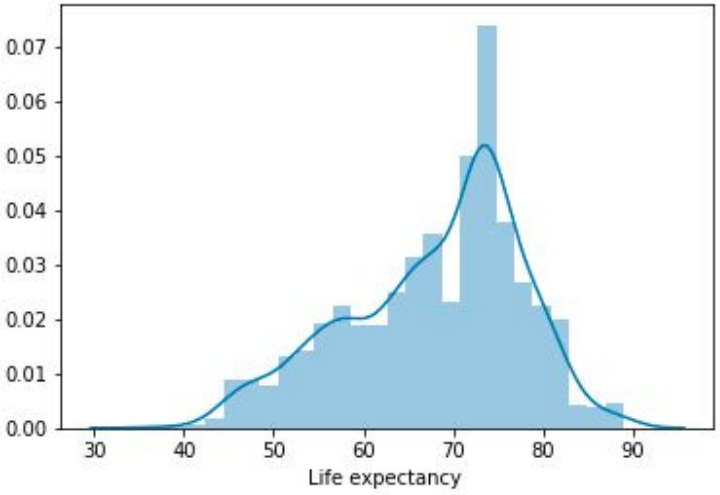
```
plt.scatter(life_data['Adult Mortality'], life_data['Life expectancy '])
plt.xlabel('Adult Mortality')
plt.ylabel('Life expectancy')
```

```
ut[62]: Text(0, 0.5, 'Life expectancy')
```

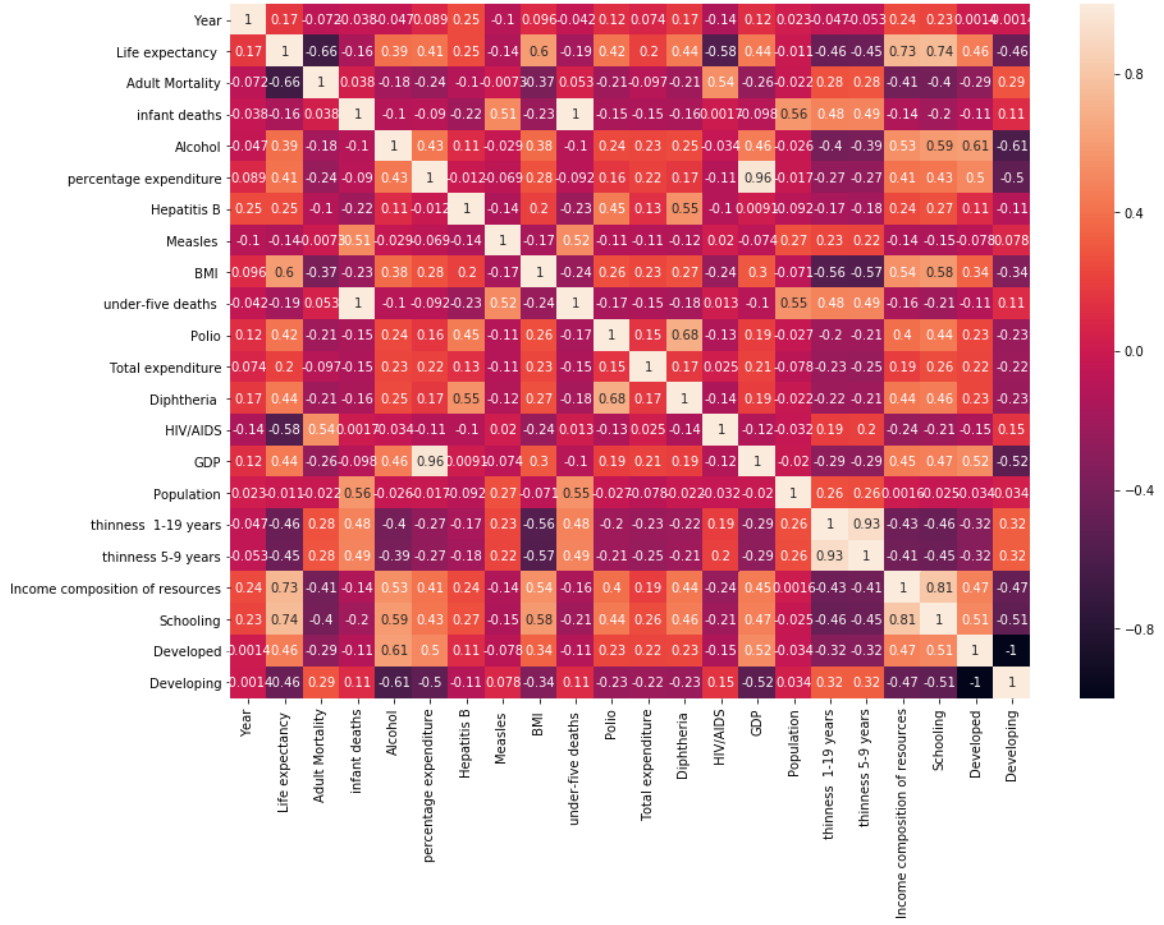



```
In [75]: sns.distplot(df_data_0['Life expectancy '])
```

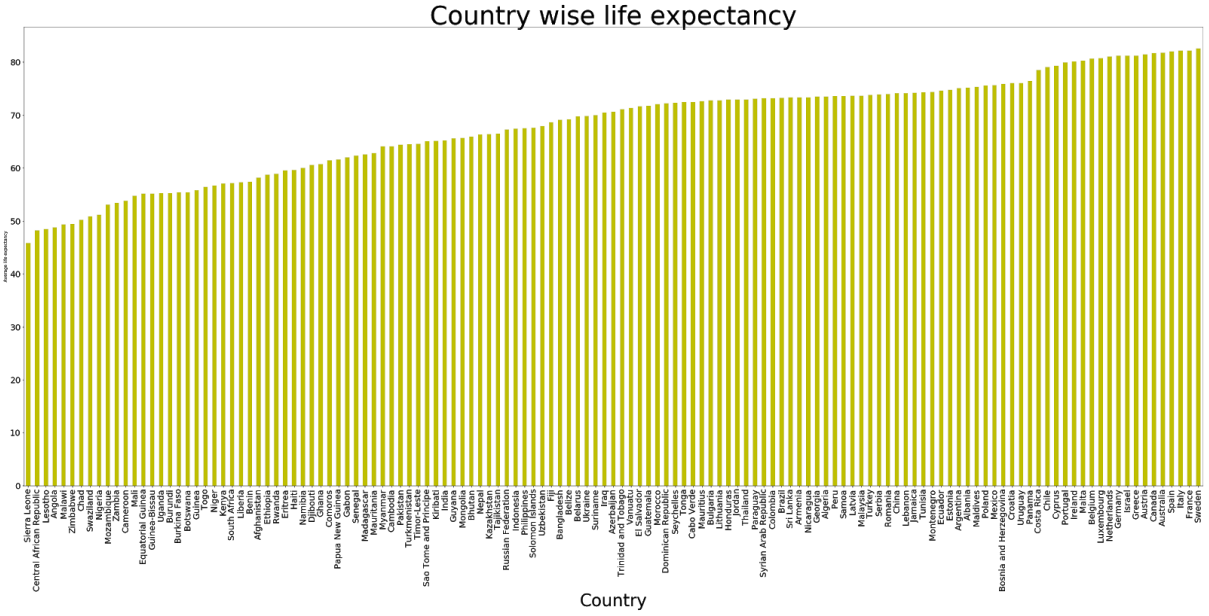
```
Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1d1bfd06a0>
```



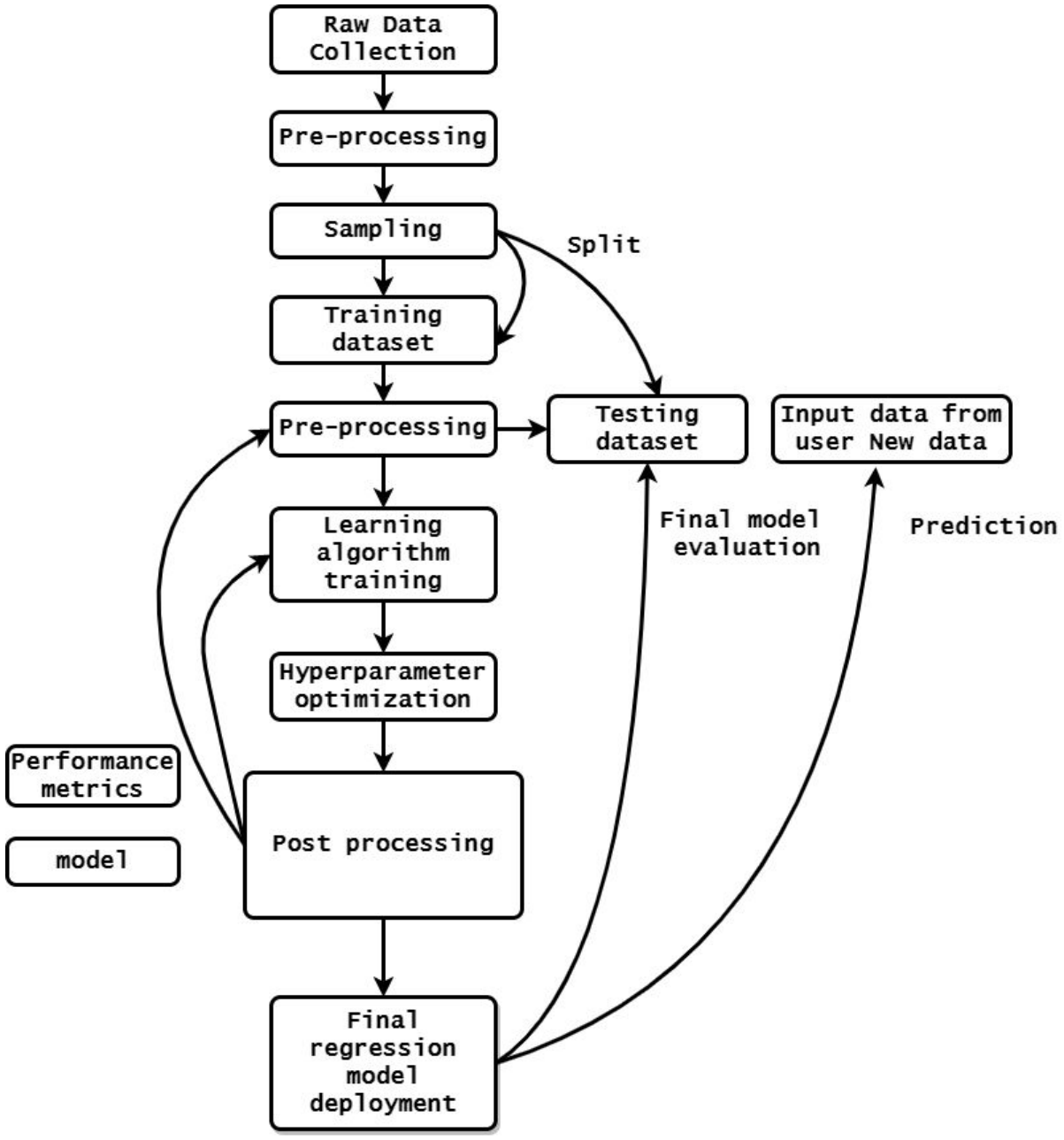
Now we will plot the correlation matrix visualizing it with a heatmap.



Country-wise life expectancy



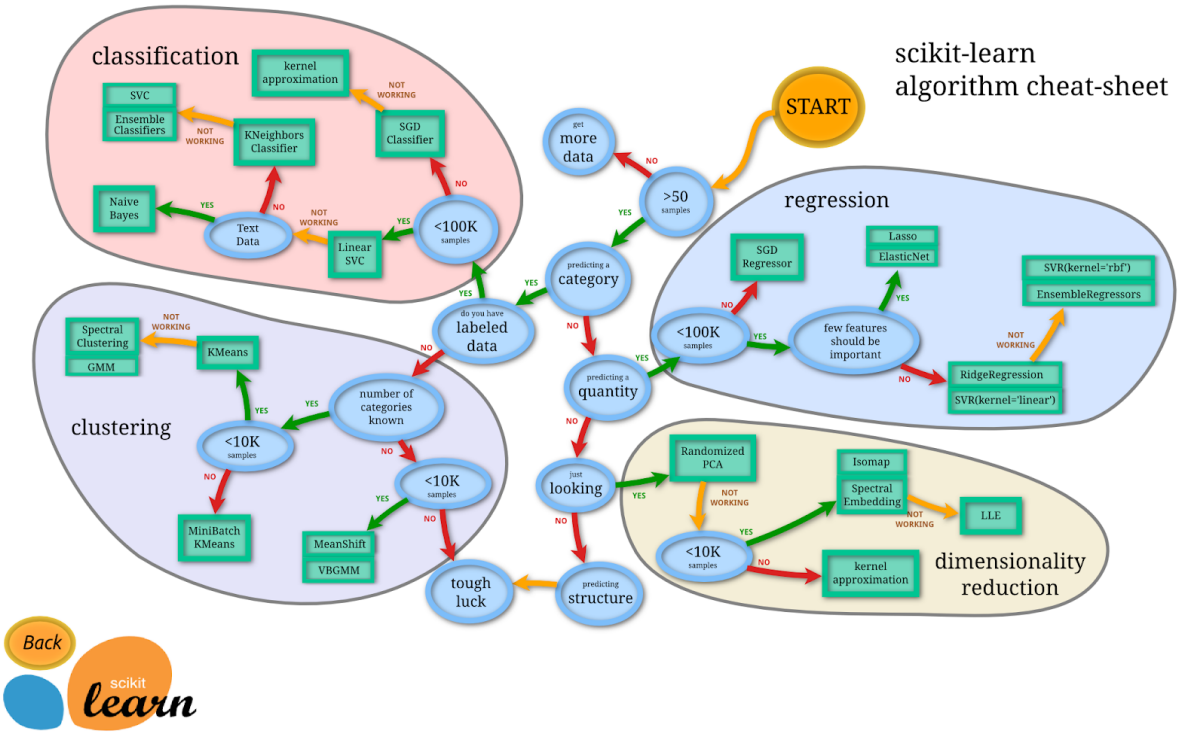
5. FLOWCHART



Flowchart for choosing the best algorithm

Often the hardest part of solving a machine learning problem can be finding the right estimator for the job.
Different estimators are better suited for different types of data and different problems.

The flowchart below is designed to give users a bit of a rough guide on how to approach problems with regard to which estimators to try on your data.



6. RESULT

Following steps were performed to obtain results

Part 1. Loading packages

The following packages have been imported: NumPy, Pandas, Matplotlib, Scipy, Seaborn. Sklearn is the most widely used package for the machine learning process. The following sub-packages have been used:

1. train_test_split
2. linear_model
3. model_selection
4. metrics
5. tree
6. ensemble
7. preprocessing

And so on.

Part 2. Reading the data

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.000000	2938.000000	2904.000000	2938.000000	291
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	80.940461	2419.592240	38.321247	42.035739	8
std	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858	25.070016	11467.272489	20.044034	160.445548	2
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	1.000000	0.000000	1.000000	0.000000	
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343	77.000000	0.000000	19.300000	0.000000	7
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906	92.000000	17.000000	43.500000	4.000000	9
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144	97.000000	360.250000	56.200000	28.000000	9
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.000000	212183.000000	87.300000	2500.000000	9

```
]: df_data_0.iloc[32]
```

```
Out[53]: Country                          Algeria
         Year                              2015
         Life expectancy                   75.6
         Adult Mortality                     19
         infant deaths                       21
         Alcohol                            NaN
         percentage expenditure               0
         Hepatitis B                         95
         Measles                             63
          BMI                              59.5
         under-five deaths                   24
         Polio                               95
         Total expenditure                  NaN
         Diphtheria                          95
          HIV/AIDS                          0.1
         GDP                            4132.76
         Population                3.98715e+07
          thinness  1-19 years                6
          thinness 5-9 years                5.8
         Income composition of resources  0.743
         Schooling                         14.4
         Developed                            0
         Developing                           1
         Name: 32, dtype: object
```

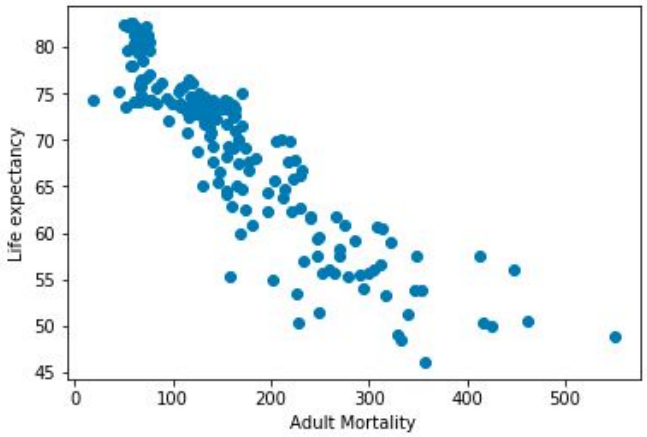
Part 3. EDA

Exploratory Data Analysis: The dataset was important in analysis, calculations were performed using python

Plotting life expectancy against adult mortality to find out the

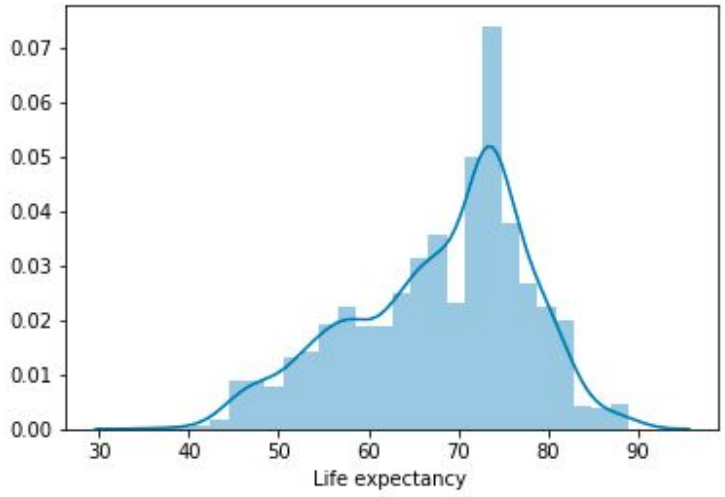
```
plt.scatter(life_data['Adult Mortality'], life_data['Life expectancy '])
plt.xlabel('Adult Mortality')
plt.ylabel('Life expectancy')
```

```
ut[62]: Text(0, 0.5, 'Life expectancy')
```

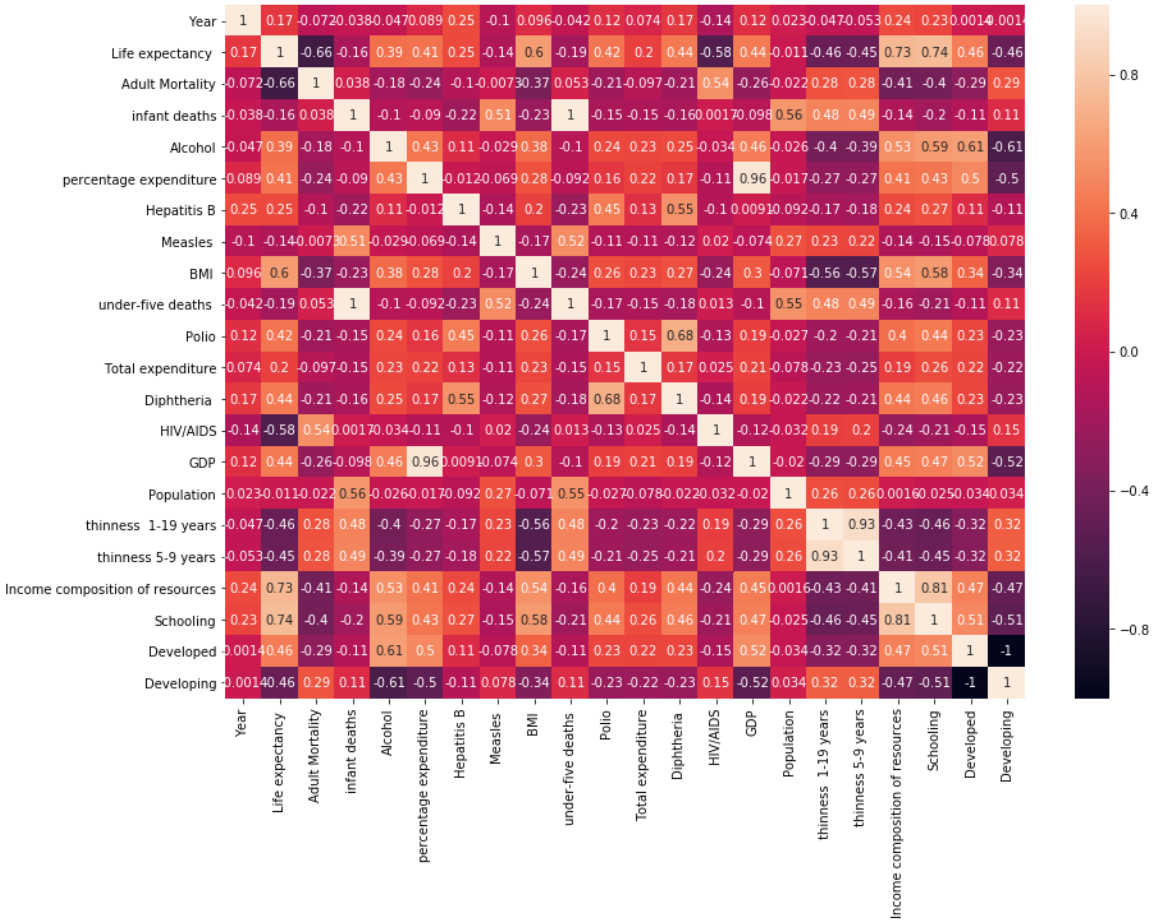



```
In [75]: sns.distplot(df_data_0['Life expectancy '])
```

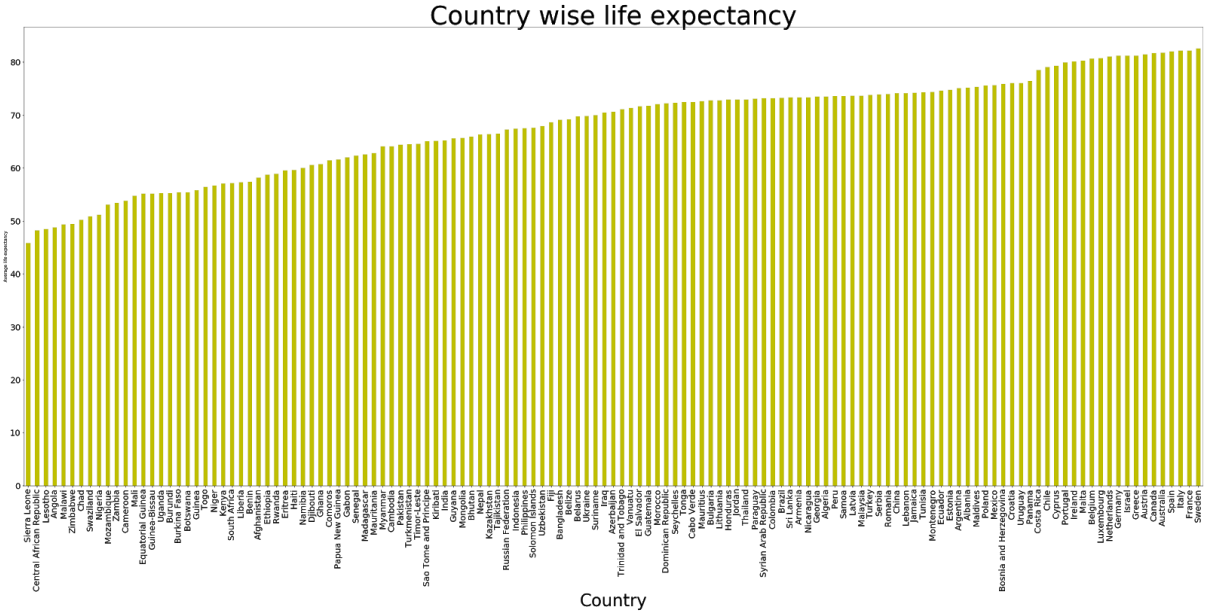
```
Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1d1bfd06a0>
```



Now we will plot the correlation matrix visualizing it with a heatmap.



Country-wise life expectancy



Part 4. Preprocessing the data

The raw data is not suitable for us to start building a model so some preprocessing will be done. First, the Status of the country is turned into numerical with the get_dummies function, so we get 2 new columns. The original column is being

dropped. Also, certain rows contain Nan values hence we group by country and try to find interpolated values.
Part 4. Splitting dataset into Training and testing. And then applying a suitable algorithm
The need is to first split up our data into an X array that contains the features to train on, and a y array with the target variable, in this case, the Life expectancy column.
The testing size is 0.3 rest is used for training.

Part 5. Machine learning model used and their score

Machine learning model used

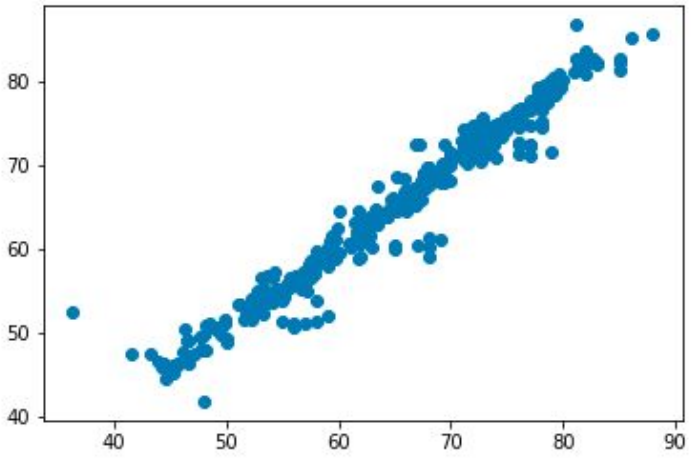
- Linear Regression
    - Linear regression is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, y can be calculated from a linear combination of the input variables (x).
    - When there is a single input variable (x), the method is referred to as simple linear regression. When there are multiple input variables, literature from statistics often refers to the method as multiple linear regression.
    - model score: 0.711
- Ridge Regression
    - Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value.
    - model score: -17997.746

- Decision Tree Regression
    - Decision trees build regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.
    - model score: 0.926
- Lasso Regression
    - Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters).
    - model score: 0.780
- Random Forest Regression
    - The random forest model is a type of additive model that makes predictions by combining decisions from a sequence of base models. A random forest is a meta-estimator (i.e. it combines the result of multiple predictions) which aggregates many decision trees.
    - model score: 0.943
- ElasticNet Regression
    - Elastic Net first emerged as a result of critique on the lasso, whose variable selection can be too dependent on data and thus unstable. The solution is to combine the penalties of ridge regression and lasso to get the best of both worlds.
    - model score: 0.781
- Extra Trees Regression
    - The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.
    - model score: 0.962

Part 6. Result

```
In [89]: predictions2=RFRegrrosor.predict(X_test)
         plt.scatter(y_test,predictions2)
```

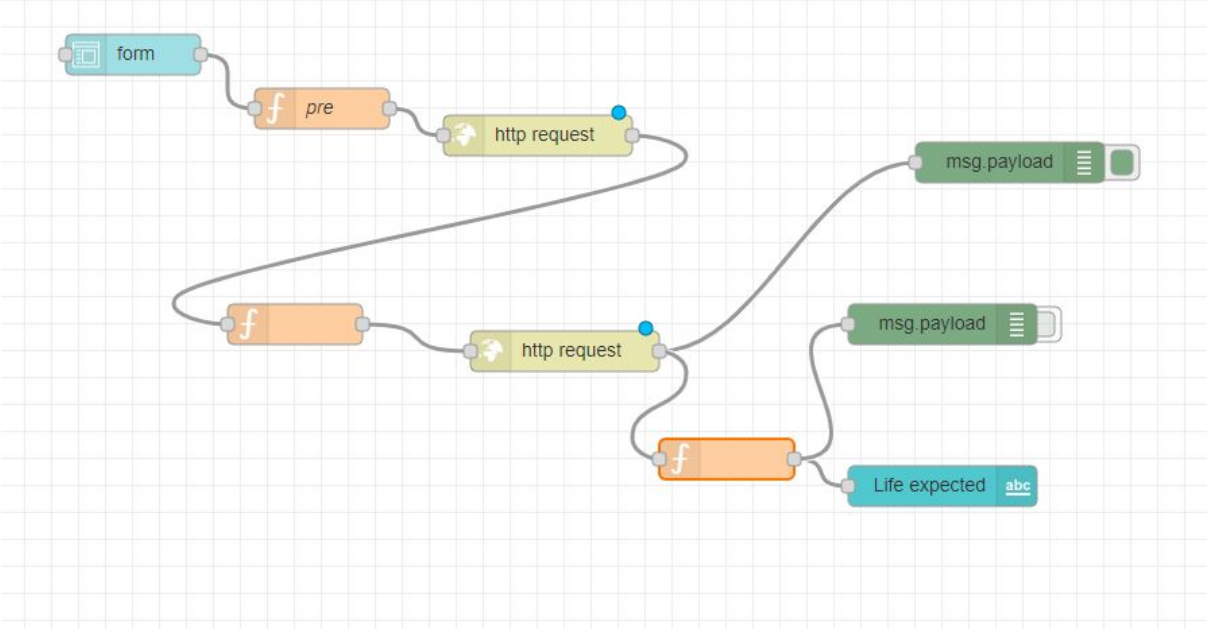
```
Out[89]: <matplotlib.collections.PathCollection at 0x7f1d16c75390>
```




```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, predictions2))
print('MSE:', metrics.mean_squared_error(y_test, predictions2))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions2)))
```

```
MAE: 0.919112227805695
MSE: 2.9533780569514243
RMSE: 1.7185395127699055
```

Node-red backend flow



Node-red front end

Machine Learning Model

Country *
India

Year *
2015

Adult Mortality *
181

infant deaths *
910

Alcohol *
3.1

percentage expenditure *
0

Hepatitis B *
87

Measles *
90387

BMI *
18.7

under-five deaths *
1100

Polio *
86

Total expenditure *
4.7

87

HIV/AIDS *

0.2

GDP *

1613.189

Population *

1395398

thinness 1-19 years *

26.7

thinness 5-9 years *

27.3

Income composition of resources *

0.615

Schooling *

11.6

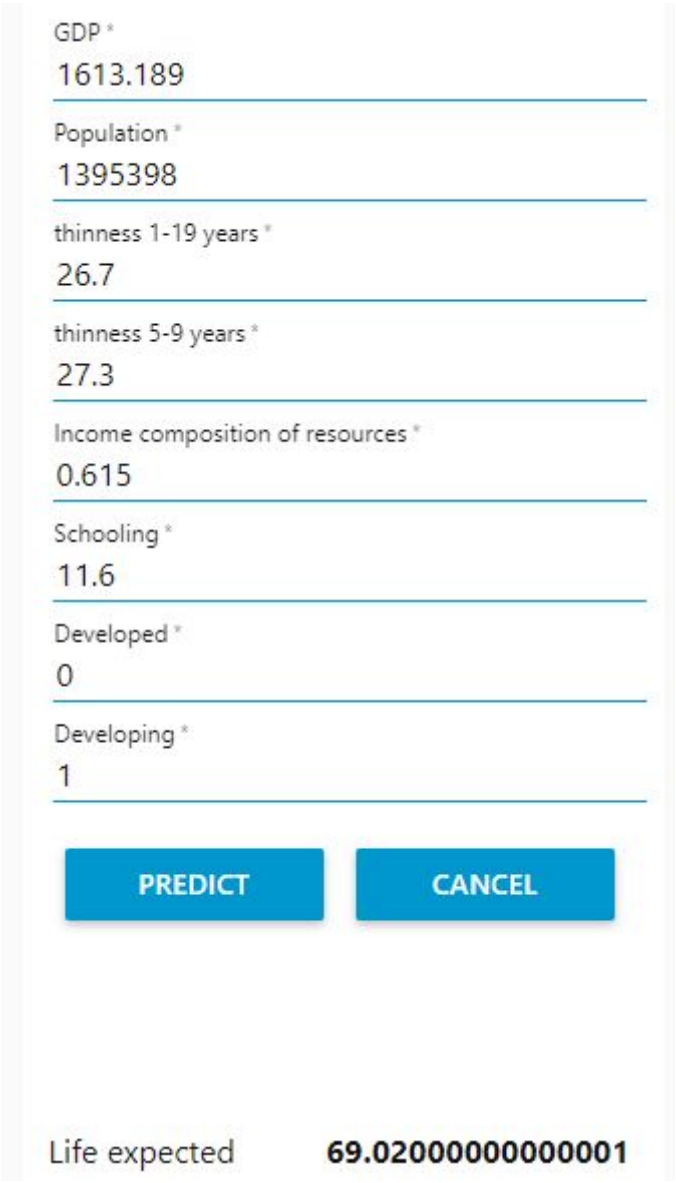
Developed *

0

Developing *

1

PREDICT CANCEL



7. ADVANTAGES & DISADVANTAGES

Advantages

- Machine learning lets you adapt without human intervention
- Automation of predicting life expectancy
- Machine Learning improves over time

Disadvantages

- The accuracy is not 100% hence the predicated results comprises of a high level of error susceptibility
- Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose.

- Machine Learning requires massive data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where they must wait for new data to be generated.

8. APPLICATIONS

- For hospitals to plan about Advance Care Planning infrastructure
- For the government to allocate a budget on the healthcare sector.
- Increasing the productivity of available resources.
- Analyzing scientific reasons behind deaths.
- Modification and customizing dietary plans for a user.
- Economic sector

9. CONCLUSION

Extra Trees Regression's model score was found to be 0.962.
This model is used for deployment which has higher accuracy as compared to other algorithms. The predicted value is displayed on the front end. Hence in this way, the life expectancy was predicted using the machine learning model and the user can input the values as per information and get the predicted value.

10. FUTURE SCOPE

- It can be applied for all living beings, different countries as a whole.
- A central point of uncertainty is whether the kind of progress and development in many countries that has marked the past couple of centuries will continue until 2050.
- Perhaps the future will be characterized by poverty, misery, and a shorter life expectancy. As people live longer, ages at death are becoming more similar.
- This dual advance over the last two centuries, a central aim of public health policies, is a major achievement of modern civilization.
- Some recent exceptions to the joint rise of life expectancy and life span equality, however, make it difficult to determine the underlying causes of this relationship.

- Here, we develop a unifying framework to study life expectancy and life span equality over time, relying on concepts about the pace and shape of ageing.
- Life Expectancy would have a major influence on the economy.
- Life expectancy inequality is forecast to continue to rise across districts, however, with present and future inequalities partly related to district deprivation and partly associated with variation within deprivation quintiles, especially within the deprived quintiles.
- Furthermore, we found that life expectancy varied more in the more deprived quintiles, perhaps because deprived communities are more vulnerable to factors that affect health and longevity, but vary somewhat independently of deprivation.
- Life expectancy is one of the most important factors in end-of-life decision making. Good prognostication, for example, helps to determine the course of treatment and helps to anticipate the procurement of health care services and facilities, or more broadly: facilitates Advance Care Planning.
- Advance Care Planning improves the quality of the final phase of life by stimulating doctors to explore the preferences for end-of-life care with their patients, and people close to the patients.

11. BIBLIOGRAPHY

i. https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(15)60296-3/fulltext
ii. https://www.pnas.org/content/117/10/5250
iii. https://medium.com/swlh/predicting-life-expectancy-w-regression-b794ca457cd4
iv. https://towardsdatascience.com/time-left-to-live-modeling-life-expectancy-and-prototyping-it-on-the-web-with-flask-and-68e3a8fa0fe4
v. https://towardsdatascience.com/what-really-drives-higher-life-expectancy-e1c1ec22f6e1
vi. https://towardsdatascience.com/columntransformer-in-scikit-for-label encoding-and-onehotencoding-in-machine-learning-c6255952731b
vii. https://www.analyticsvidhya.com/blog/2020/01/build-your-first-machine-learning-pipeline-using-scikit-learn/
viii. https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

ix. https://towardsdatascience.com/how-to-create-your-first-machine-learning-model-4c8f745e4b8c
x. https://towardsdatascience.com/linear-regression-using-python-b136c91bf0a2
xi. https://www.datasciencesociety.net/using-machine-learning-to-explain-and-predict-the-life-expectancy-of-different-countries/

12. APPENDIX

i. Source code

```
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

#@hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes
your credentials.
# You might want to remove those credentials before you share the notebook.
client_883fcc5cd8954037a8949c3cb785b2c6 = ibm_boto3.client(service_name='s3',
   ibm_api_key_id="**********************************************",
   ibm_auth_endpoint="**********************************************",
   config=Config(signature_version='oauth'),
   endpoint_url="**********************************************")

body =
client_883fcc5cd8954037a8949c3cb785b2c6.get_object(Bucket='lifeexpectancy-don
otdelete-pr-clqlzr9exlickh',Key='datasets_12603_17232_Life Expectancy Data
(1).csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

# If you are reading an Excel file into a pandas DataFrame, replace `read_csv` by
`read_excel` in the next statement.
df_data_0 = pd.read_csv(body)
df_data_0.head()

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline
```

```
"""### Check out the Data"""

df_data_0.head()

status = pd.get_dummies(df_data_0.Status)

df_data_0.info()

df_data_0.describe()

df_data_0 = pd.concat([df_data_0, status], axis = 1)
df_data_0

df_data_0 = df_data_0.drop(['Status'], axis=1)

df_data_0

df_data_0.iloc[32]

df_data_0.iloc[33]

life_data = df_data_0.groupby('Country').mean()

life_data.head()

life_data.columns

plt.scatter(life_data[' HIV/AIDS'], life_data['Life expectancy '])
plt.xlabel('HIV/AIDS')
plt.ylabel('Life expectancy')

plt.scatter(life_data[' BMI '], life_data['Life expectancy '])
plt.xlabel('BMI')
plt.ylabel('Life expectancy')

plt.scatter(life_data['under-five deaths '], life_data['Life expectancy '])
plt.xlabel('under-five deaths')
plt.ylabel('Life expectancy')

plt.scatter(life_data['Alcohol'], life_data['Life expectancy '])
plt.xlabel('Alcohol')
plt.ylabel('Life expectancy')

plt.scatter(life_data['Adult Mortality'], life_data['Life expectancy '])
plt.xlabel('Adult Mortality')
plt.ylabel('Life expectancy')

plt.scatter(life_data['Schooling'], life_data['Life expectancy '])
plt.xlabel('Schooling')
plt.ylabel('Life expectancy')
```

```
plt.scatter(life_data['percentage expenditure'], life_data['Life expectancy '])
plt.xlabel('Percentage Healhcare expenditure')
plt.ylabel('Life expectancy')

df_data_0.iloc[32]

df_data_0.describe()

country_list=df_data_0.Country.unique()
print(len(country_list))
country_list

df_data_0.columns

for countr in country_list:

df_data_0.loc[df_data_0['Country']==countr,df_data_0.columns]=df_data_0.loc[df_data
_0['Country']==countr,df_data_0.columns].interpolate()
df_data_0.dropna(inplace=True)

df_data_0

df_data_0.isnull().values.any()

df_data_0.shape

df_data_0.columns


sns.pairplot(df_data_0)

sns.distplot(df_data_0['Life expectancy '])

plt.figure(figsize = (14, 10))
sns.heatmap(df_data_0.corr(),annot = True)
life_data = life_data.drop('Year', axis = 1)

plt.figure(figsize = (14, 10))
sns.heatmap(life_data.corr(), annot = True)

import pandas as pd
X = df_data_0[[ 'Year', 'Adult Mortality',
      'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
      'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
      'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
      ' thinness  1-19 years', ' thinness 5-9 years',
      'Income composition of resources', 'Schooling', 'Developed',
      'Developing']]
y = pd.DataFrame(data=df_data_0,columns=['Life expectancy '])

X.head()
```

```
y.head()

## Train Test Split


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
from sklearn.metrics import accuracy_score, log_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
GradientBoostingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

from sklearn.ensemble import ExtraTreesRegressor
models = [
  LinearRegression(),
   ExtraTreesRegressor(),
   Ridge(),
   DecisionTreeRegressor(),
   Lasso(),
   RandomForestRegressor(),
   ElasticNet()
   ]
for modeler in models:
   model = modeler.fit(X_train, y_train)
   score=r2_score(model.predict(X_test),y_test)
   print(modeler)
   print("model score: %.3f" % score)

''' Training a Regression Model by preprocessing it and including both ##numeric and
non numeric values

Let's now begin to train out regression model! We will need to first split up our data
into an X array that contains the features to train on, and a y array with the target
variable, in this case the Life expectancy column.

### X and y arrays
'''


import pandas as pd
X = df_data_0[[ 'Country','Year', 'Adult Mortality',
      'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
      'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
      'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
      ' thinness  1-19 years', ' thinness 5-9 years',
      'Income composition of resources', 'Schooling', 'Developed',
```

```
    'Developing']]
y = pd.DataFrame(data=df_data_0,columns=['Life expectancy '])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)


from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
num_feat=['Year', 'Adult Mortality',
    'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
    'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
    'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
    ' thinness  1-19 years', ' thinness 5-9 years',
    'Income composition of resources', 'Schooling', 'Developed',
    'Developing']
cat_feat=['Country']
categorical_transformer=Pipeline(steps=[('onehot',OneHotEncoder(handle_unknown='
ignore'))])

numeric_transformer=Pipeline(steps=[('imputer',SimpleImputer(strategy='median'))])

from sklearn.compose import ColumnTransformer
preprocessor =
ColumnTransformer(transformers=[('cat',categorical_transformer,cat_feat),('num',nu
meric_transformer,num_feat)])

from collections import OrderedDict
modeling=OrderedDict([("Linear
Regression",Pipeline([('preprocessor',preprocessor),('RFRegressor',LinearRegression()
)])),
                ("Extra Trees
Regression",Pipeline([('preprocessor',preprocessor),('RFRegressor',ExtraTreesRegress
or())])),
                ("Ridge
Regression",Pipeline([('preprocessor',preprocessor),('RFRegressor', Ridge())])),
                ("Decision Tree
Regression",Pipeline([('preprocessor',preprocessor),('RFRegressor',DecisionTreeRegre
ssor())])),
                ("Lasso
Regression",Pipeline([('preprocessor',preprocessor),('RFRegressor',Lasso())])),
                ("Random Forest
Regression",Pipeline([('preprocessor',preprocessor),('RFRegressor',RandomForestReg
ressor())])),
                ("ElasticNet
Regression",Pipeline([('preprocessor',preprocessor),('RFRegressor',ElasticNet())]))



            ])
```

```
for (name,modelers) in modeling.items():
    modelers.fit(X_train, y_train)
    scoree=r2_score(modelers.predict(X_test),y_test)
    print(name)
    print("model score: %.3f" % scoree)

"""## Extra Trees Regression yeilds the highest score so we use to deploy it"""

RFRegrrosor=Pipeline([('preprocessor',preprocessor),('RFRegressor',ExtraTreesRegres
sor())])

RFRegrrosor.fit(X_train, y_train)

"""## Predictions from our Model
Let's grab predictions off our test set and see how well it did!
"""

predictions2=RFRegrrosor.predict(X_test)
plt.scatter(y_test,predictions2)

"""## Regression Evaluation Metrics


Here are three common evaluation metrics for regression problems:

**Mean Absolute Error** (MAE) is the mean of the absolute value of the errors:

$$\frac 1n\sum_{i=1}^n|y_i-\hat{y}_i|$$

**Mean Squared Error** (MSE) is the mean of the squared errors:

$$\frac 1n\sum_{i=1}^n(y_i-\hat{y}_i)^2$$

**Root Mean Squared Error** (RMSE) is the square root of the mean of the squared
errors:

$$\sqrt{\frac 1n\sum_{i=1}^n(y_i-\hat{y}_i)^2}$$

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which
tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y"
units.

All of these are **loss functions**, because we want to minimize them.
"""

from sklearn import metrics
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions2))
print('MSE:', metrics.mean_squared_error(y_test, predictions2))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions2)))

life_labels = life_data['Life expectancy ']
life_features = life_data.drop('Life expectancy ', axis = 1)

life_features.isnull().head()

life_features.isnull().sum()

life_labels.isnull().sum()

life_features.fillna(value = life_features.mean(), inplace = True)

life_labels.isnull().sum()

life_features

life_features.isnull().head()

life_labels.fillna(value = life_labels.mean(), inplace = True)

life_features.head()

life_features.iloc[1]

life_labels.iloc[1]

from scipy import stats
stats.describe(life_features[1:])

from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()

life_features = min_max_scaler.fit_transform(life_features)

life_features

from sklearn.model_selection import train_test_split
life_features_train, life_features_test, life_labels_train, life_labels_test =
train_test_split( life_features, life_labels, train_size = 0.7, test_size = 0.3)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from nose.tools import *
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.metrics import make_scorer
from scipy import stats
import seaborn as sns
linear_model = LinearRegression()
linear_model.fit(life_features_train, life_labels_train)

print('R_square score on the training: %.2f' % linear_model.score(life_features_train,
life_labels_train))

linear_model_predict = linear_model.predict(life_features_test)

life_features_test

# Commented out IPython magic to ensure Python compatibility.
print('Coefficients: \n', linear_model.coef_)
print("Mean squared error: %.2f"
#       % mean_squared_error(life_labels_test, linear_model_predict))
print("Mean absolute error: %.2f"
#       % mean_absolute_error(life_labels_test, linear_model_predict))
print('R_square score: %.2f' % r2_score(life_labels_test, linear_model_predict))

"""## Hyperparameter optimisation
Each machine learning algorithm has a wide number of parameters that are used to
control the learning process. These parameters can be changed and depending on
the data set can result in an increase in performance for the model. The process of
finding the best set of parameters for an algorithm and data set is known as
hyperparameter optimisation.

## Hyperparameter optimisation using Ridge regressor
"""

# Commented out IPython magic to ensure Python compatibility.
scoring = make_scorer(r2_score)
grid_cv = GridSearchCV(Ridge(),
            param_grid={'alpha': range(0, 10), 'max_iter' : [10, 100, 1000]},
            scoring=scoring, cv=5, refit=True)
grid_cv.fit(life_features_train, life_labels_train)
print("Best Parameters: " + str(grid_cv.best_params_))
result = grid_cv.cv_results_
print("R^2 score on training data: %.2f" %grid_cv.score(life_features_train,
life_labels_train))
```

```
print("R^2 score: %.2f"
#       % r2_score(life_labels_test, grid_cv.best_estimator_.predict(life_features_test)))
print("Mean squared error: %.2f"
#       % mean_squared_error(life_labels_test, linear_model_predict))
print("Mean absolute error: %.2f"
#       % mean_absolute_error(life_labels_test, linear_model_predict))

"""## Hyperparameter optimisation using Lasso regressor"""

# Commented out IPython magic to ensure Python compatibility.
scoring = make_scorer(r2_score)
grid_cv = GridSearchCV(Lasso(),
            param_grid={'alpha': range(0, 10), 'max_iter' : [10, 100, 1000]},
            scoring=scoring, cv=5, refit=True)
grid_cv.fit(life_features_train, life_labels_train)
print("Best Parameters: " + str(grid_cv.best_params_))
result = grid_cv.cv_results_
print("R^2 score on training data: %.2f" % grid_cv.score(life_features_train,
life_labels_train))
print("R^2 score: %.2f"
#       % r2_score(life_labels_test, grid_cv.best_estimator_.predict(life_features_test)))
print("Mean squared error: %.2f"
#       % mean_squared_error(life_labels_test, linear_model_predict))
print("Mean absolute error: %.2f"
#       % mean_absolute_error(life_labels_test, linear_model_predict))

"""## Hyperparameter optimisation using ElasticNet regressor"""

# Commented out IPython magic to ensure Python compatibility.
scoring = make_scorer(r2_score)
grid_cv = GridSearchCV(ElasticNet(),
            param_grid={'alpha': range(0, 10), 'max_iter' : [10, 100, 1000], 'l1_ratio' : [0.1,
0.4, 0.8]},
            scoring=scoring, cv=5, refit=True)
grid_cv.fit(life_features_train, life_labels_train)
print("Best Parameters: " + str(grid_cv.best_params_))
result = grid_cv.cv_results_
print("R^2 score on training data: %.2f" % grid_cv.score(life_features_train,
life_labels_train))
print("R^2 score: %.2f"
#       % r2_score(life_labels_test, grid_cv.best_estimator_.predict(life_features_test)))
print("Mean squared error: %.2f"
#       % mean_squared_error(life_labels_test, linear_model_predict))
print("Mean absolute error: %.2f"
#       % mean_absolute_error(life_labels_test, linear_model_predict))

"""## Linear Regression with Polynomial Features"""

quad_feature_transformer = PolynomialFeatures(2, interaction_only = True)
quad_feature_transformer.fit(life_features_train)
life_features_train_quad = quad_feature_transformer.transform(life_features_train)
```

```
life_features_test_quad = quad_feature_transformer.transform(life_features_test)

poly_model_quad = LinearRegression()
poly_model_quad.fit(life_features_train_quad, life_labels_train)
accuracy_score_quad = poly_model_quad.score(life_features_train_quad,
life_labels_train)
print(accuracy_score_quad)

poly_model_quad_predict = poly_model_quad.predict(life_features_test_quad)

# Commented out IPython magic to ensure Python compatibility.
print("Mean squared error: %.2f"
#       % mean_squared_error(life_labels_test, poly_model_quad_predict))
print("Mean absolute error: %.2f"
#       % mean_absolute_error(life_labels_test, poly_model_quad_predict))
print('R_square score: %.2f' % r2_score(life_labels_test, poly_model_quad_predict))

"""All the errors are significantly higher then the privious models, and the R square is
in this case negative (this happens only in sklearn). This is the worst performing
model for now.

## Hyperparameter optimisation using Decision Tree Regression
"""

decision_tree_model = DecisionTreeRegressor()
decision_tree_fit = decision_tree_model.fit(life_features_train, life_labels_train)
decision_tree_score = cross_val_score(decision_tree_fit, life_features_train,
life_labels_train, cv = 5)

print("mean cross validation score: %.2f"  % np.mean(decision_tree_score))
print("score without cv: %.2f" % decision_tree_fit.score(life_features_train,
life_labels_train))
print("R^2 score on the test data %.2f"% r2_score(life_labels_test,
decision_tree_fit.predict(life_features_test)))

decision_tree_model_predict = decision_tree_model.predict(life_features_test)

# Commented out IPython magic to ensure Python compatibility.
scoring = make_scorer(r2_score)
grid_cv = GridSearchCV(DecisionTreeRegressor(),
          param_grid={'min_samples_split': range(2, 10)},
          scoring=scoring, cv=5, refit=True)
grid_cv.fit(life_features_train, life_labels_train)
grid_cv.best_params_

print("Best Parameters: " + str(grid_cv.best_params_))
result = grid_cv.cv_results_
print("R^2 score on training data: %.2f"  %
grid_cv.best_estimator_.score(life_features_train, life_labels_train))
print("R^2 score: %.2f"
#       % r2_score(life_labels_test, grid_cv.best_estimator_.predict(life_features_test)))
```

```
print("Mean squared error: %.2f"
#       % mean_squared_error(life_labels_test, decision_tree_model_predict))
print("Mean absolute error: %.2f"
#       % mean_absolute_error(life_labels_test, decision_tree_model_predict))

"""## Hyperparameter optimisation using Random Forest Regression"""

# Commented out IPython magic to ensure Python compatibility.
random_forest_model = RandomForestRegressor()
random_forest_fit = random_forest_model.fit(life_features_train, life_labels_train)
random_forest_score = cross_val_score(random_forest_fit, life_features_train,
life_labels_train, cv = 5)
print("mean cross validation score: %.2f"
#       % np.mean(random_forest_score))
print("score without cv: %.2f"
#       % random_forest_fit.score(life_features_train, life_labels_train))
print("R^2 score on the test data %.2f"
#       %r2_score(life_labels_test, random_forest_fit.predict(life_features_test)))

random_forest_model_predict = random_forest_model.predict(life_features_test)

# Commented out IPython magic to ensure Python compatibility.
scoring = make_scorer(r2_score)
grid_cv = GridSearchCV(RandomForestRegressor(),
            param_grid={'min_samples_split': range(2, 10)},
            scoring=scoring, cv=5, refit=True)
grid_cv.fit(life_features_train, life_labels_train)
grid_cv.best_params_
result = grid_cv.cv_results_
print("Best Parameters: " + str(grid_cv.best_params_))
result = grid_cv.cv_results_
print("R^2 score on training data: %.2f"  %
grid_cv.best_estimator_.score(life_features_train, life_labels_train))
print("R^2 score: %.2f"
#       % r2_score(life_labels_test, grid_cv.best_estimator_.predict(life_features_test)))
print("Mean squared error: %.2f"
#       % mean_squared_error(life_labels_test, random_forest_model_predict))
print("Mean absolute error: %.2f"
#       % mean_absolute_error(life_labels_test, random_forest_model_predict))

"""## Hyperparameter optimisation using Extra Trees Regressor"""

# Commented out IPython magic to ensure Python compatibility.
from sklearn.ensemble import ExtraTreesRegressor

life_features_train, life_features_test, life_labels_train, life_labels_test =
train_test_split( life_features, life_labels, train_size = 0.7, test_size = 0.3)
reg = ExtraTreesRegressor(n_estimators=100, random_state=0).fit(
life_features_train, life_labels_train)
reg.score(life_features_test, life_labels_test)
print("mean reg score with random state-0: %.2f"
```

```
#       % reg.score(life_features_test, life_labels_test))
ExtraTreesRegressor_model = ExtraTreesRegressor(n_estimators=1000,
random_state=50)
ExtraTreesRegressor_fit = ExtraTreesRegressor_model.fit(life_features_train,
life_labels_train)
ExtraTreesRegressor_score = cross_val_score(ExtraTreesRegressor_fit,
life_features_train, life_labels_train, cv = 5)
print("mean cross validation score: %.2f"
#       % np.mean(ExtraTreesRegressor_score))
print("score without cv: %.2f"% reg.score(life_features_test, life_labels_test))
print("R^2 score on the test data %.2f"%r2_score(life_labels_test,
ExtraTreesRegressor_fit.predict(life_features_test)))


print("mean cross validation score: %.2f"
#       % np.mean(ExtraTreesRegressor_score))
print("score without cv: %.2f"
#      % ExtraTreesRegressor_fit.score(life_features_test, life_labels_test))
print("R^2 score on the test data %.2f"
#      %r2_score(life_labels_test, ExtraTreesRegressor_fit.predict(life_features_test)))

# Commented out IPython magic to ensure Python compatibility.
from sklearn.ensemble import ExtraTreesRegressor
grid_cv = GridSearchCV(ExtraTreesRegressor(),
param_grid={'min_samples_split': range(2, 10)},
            scoring=scoring, cv=5, refit=True)
grid_cv.fit(life_features_train, life_labels_train)
grid_cv.best_params_

print("Best Parameters: " + str(grid_cv.best_params_))
result = grid_cv.cv_results_
print("R^2 score on training data: %.2f"  %
grid_cv.best_estimator_.score(life_features_train, life_labels_train))
print("R^2 score: %.2f"
#      % r2_score(life_labels_test, grid_cv.best_estimator_.predict(life_features_test)))
print("Mean squared error: %.2f"
#      % mean_squared_error(life_labels_test, decision_tree_model_predict))
print("Mean absolute error: %.2f"
#      % mean_absolute_error(life_labels_test, decision_tree_model_predict))

!pip install watson-machine-learning-client

from watson_machine_learning_client import WatsonMachineLearningAPIClient

# @hidden_cell
wml_credentials={
  "apikey": "*******************************************",
  "instance_id": "*******************************************",
 "url": "https://eu-gb.ml.cloud.ibm.com"
}
```

```
client = WatsonMachineLearningAPIClient( wml_credentials )

# @hidden_cell
model_props = {client.repository.ModelMetaNames.AUTHOR_NAME: "Neeraj",
            client.repository.ModelMetaNames.AUTHOR_EMAIL:
"Neerajpatil14@gmail.com",
            client.repository.ModelMetaNames.NAME: "Lifeexpectancy2"}

model_artifact =client.repository.store_model(RFRegrrosor,
meta_props=model_props)

published_model_uid = client.repository.get_model_uid(model_artifact)

# @hidden_cell\
published_model_uid

# @hidden_cell
deployment = client.deployments.create(published_model_uid, name="Life
expectancy2")

scoring_endpoint = client.deployments.get_scoring_url(deployment)

# @hidden_cell
scoring_endpoint
```