INSTITUTE FOR ADVANCED
COMPUTING AND
SOFTWARE
DEVELOPMENT
AKURDI, PUNE

Documentation On

"Life Expectancy Prediction"

PG-DBDA MAR 2022

Submitted By:

Group No: 9

Shantanu Dixit 223317
Namrata Dhane 223330

Mr. Prashant Karhale
Centre Coordinator

Mr. Akshay Tilekar
Project Guide

Contents

LIST OF FIGURES

1. Introduction

1.1 Problem Statement

Life Expectancy Prediction

1.2 Abstract

Life expectancy models have vast effects on the social and financial structures of many countries around the world. Many studies have suggested the essential implications of Life expectancy predictions on social aspects and healthcare system management around the globe. These models provide many ways to improve healthcare and advanced care planning mechanism related to society. However, with time, it was observed that many present determinants were not enough to predict the longevity of the generic set of population.

Previous models were based upon mortality-based knowledge of the targeted sampling population. With the advancement in forecasting technologies and rigorous work of the past, individuals have proposed this fact that other than mortality rate, there are still many factors needed to be addressed in order to deduce the standard Predicted Life Expectancy Models. Therefore, the aim of this project is to predict life expectancy using different machine learning algorithms and have achieved better accuracy based on pertinent features of the dataset.

1.3 Aims & Objectives

The primary goal of this project is to extract quality patterns from the World Development Indicator (WDI) provided by the world bank, the non-communicable mortality rate, the suicide rate and the number of health workforce data by the World Health Organization (WHO) dataset and use the trained models to predict life expectancy. As we have data of 60 years. After the training and testing is done the Regression models will be compared based on their mean squared error (MSE) , mean absolute error (MAE) and R2 score.

2. Overall Description

2.1 Workflow of Project:

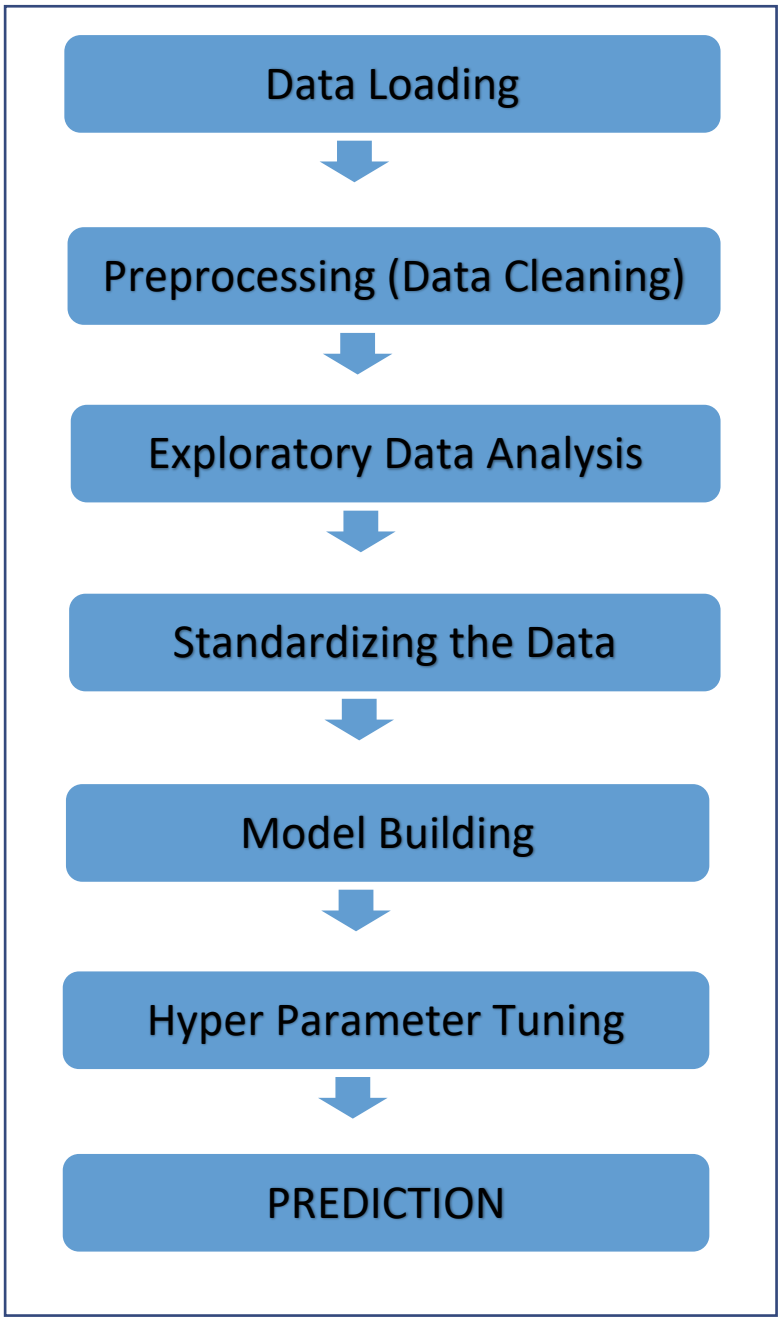

Fig 2.1 Workflow of the Project

2.2 Data Preprocessing and Cleaning:

The data can have many irrelevant, NA & NAN values & missing parts. To handle this part, data cleaning is done.

2.2.1 : Pivoting and Unpivoting:

Converted year column to rows using melt function and further as per necessity reconverted rows to year after merging few columns using pivot_table function.

```
## Lets transform the year from columns to one row and its value to value column.
wdi_data = pd.melt(wdi_data, id_vars = ["Country Name", "Country Code", "Indicator Name", "Indicator Code"],
                   var_name = "Year", value_name = "Value")
```

Fig 2.2.1.1 Unpivoting (Melting)

```
## pivot the Indicator Code to column and transform the value based on counrty name, country code and year.
wdi_data = wdi_data.pivot_table(values = 'Value', index = ['Country Name', 'Country Code', 'Year'],
                                columns = ['Indicator Code']).reset_index()
wdi_data.columns.name = None
```

Fig 2.2.1.2 Pivoting

2.2.2 : Merging Dataframes:

After transforming all columns into its respective dataframes and renaming them into human readable format we merged all dataframes together to create single dataframe.

```
sucide_rate['Year'] = sucide_rate['Year'].astype('int')
merge_dataset = merge_dataset.merge(sucide_rate, on = ["Country", "Year", "Gender"], how = 'left')
merge_dataset
```

Fig 2.2.2.1 Merging

2.2.3 Fill missing values:

We had few missing values in our data. We imputed the missing values by mean after grouping the countries.

```
def replaceByMean(colname):
    before_replace = merge_datasetSub[colname].isna().sum()
    print(before_replace)

    ## replace missing values by mean after grouping the countries
    merge_datasetSub[colname] = merge_datasetSub.groupby("Country")[colname].transform(lambda x: x.fillna(x.mean()))

    after_replace = merge_datasetSub[colname].isna().sum()
    print(after_replace)
```

Fig 2.2.3.1 Fill missing values

2.2.4 Filtration:

After cleaning data, we observed that most of the data are missing before 2000. Hence, we took a subset of the data between 2000 and 2019.

```
merge_datasetSub = merge_dataset[merge_dataset['Year'] >= 2000]
```

Fig 2.2.4.1 Filtration

2.3 Exploratory Data Analysis (EDA):

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

STEPS PERFORMED:

1. **Numerical column description**: With this we got to know count of not-empty value, average mean, standard deviation, minimum & maximum value and 25%, 50% & 75% values.

	Year	Life expectancy	Unemployment	Infant Mortality	GDP	GNI	Clean fuels and cooking technologies	Per Capita	Mortality caused by road traffic injury	Tuberculosis Incidence	DP Immunizatio
count	9928.000	9928.000	9928.000	9928.000	9928.000	9928.000	9928.000	9928.000	9928.000	9928.000	9928.00
mean	2009.495	69.834	8.256	30.235	2032870320199.638	2042053577025.779	60.593	12668.853	18.229	136.043	85.68
std	5.762	9.302	6.155	25.461	7249369673698.859	7254721460142.498	36.964	19447.598	8.223	176.090	14.20
min	2000.000	38.861	0.052	1.400	63101272.370	72802185.001	0.000	111.927	0.000	0.000	19.00
25%	2005.000	63.894	4.331	9.300	7267552276.129	7584112233.046	23.000	1366.059	12.700	21.000	81.00
50%	2009.000	71.445	6.561	23.300	45247974037.038	45940355554.258	61.950	4348.605	18.229	82.000	90.10
75%	2014.000	76.845	10.133	44.865	501278724553.924	523061711019.377	99.334	14443.944	24.100	159.000	96.00
max	2019.000	88.100	43.166	146.200	87568054407493.094	87532722519257.594	100.000	180366.715	64.600	1590.000	99.00

Fig 2.3.1 Column Description

2. **Skewness**: We checked for skewness value of numerical columns.

```
life_exep_data.skew(axis = 0, numeric_only = True)
```

Year	-0.001
Life expectancy	-0.617
Unemployment	1.768
Infant Mortality	1.089
GDP	5.889
GNI	5.910
Clean fuels and cooking technologies	-0.383
Per Capita	2.994
Mortality caused by road traffic injury	0.213
Tuberculosis Incidence	3.097
DPT Immunization	-1.661
HepB3 Immunization	-1.792
Measles Immunization	-1.434
Hospital beds	1.668
Basic sanitation services	-0.759
Tuberculosis treatment	-2.230
Urban population	0.067
Rural population	-0.067
Non-communicable Mortality	0.880
Sucide Rate	3.686

dtype: float64

Fig 2.3.2 Skewness

3. **Histogram**: This is performed to check the frequency distribution and for normality of data of all columns. Suicide Rate data is skewed towards right.

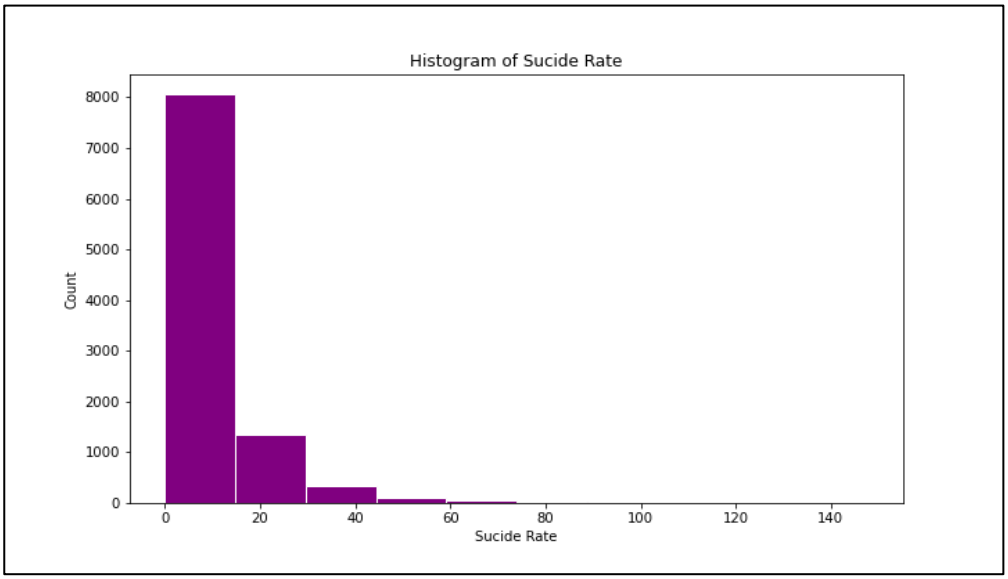

Fig 2.3.3 Histogram

4. **Box Plot**: This is done to check for outliers in the columns, inter-quartile range, median values. There are many columns with outliers.

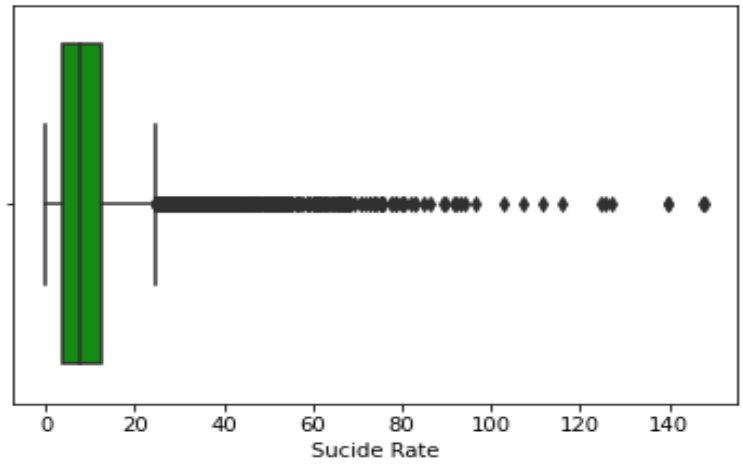

Fig 2.3.4.1 Box Plot

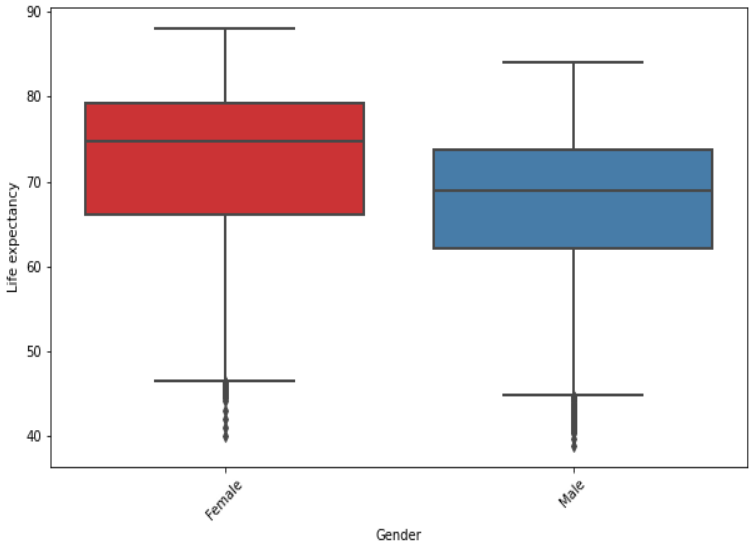

Fig 2.3.4.2 Box Plot

From the above boxplot, we can easily predict that female has higher Life expectancy than male.

5. **Outlier detection and imputation**: Outlier Management is the science of investigating and applying a suitable treatment to the outliers in the data. After exploratory data analysis, we got to know there were outliers in the dataset which needs to be managed. We have imputed the outliers to the max and min value of that column.

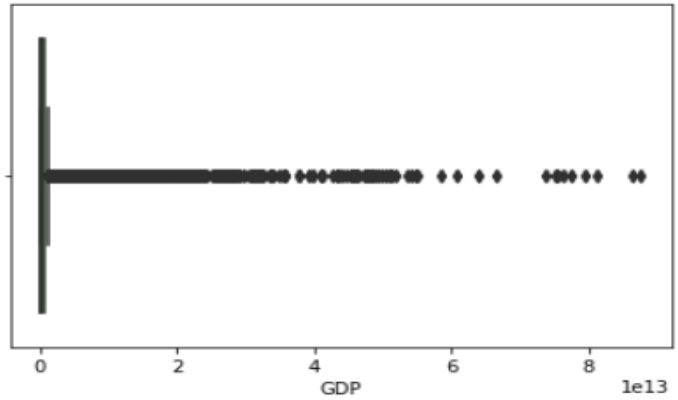

Fig 2.3.5.1 Outlier Detection

```
life_exep_data['GDP'] = np.log(life_exep_data['GDP'])
```

Fig 2.3.5.2 Outlier Imputation

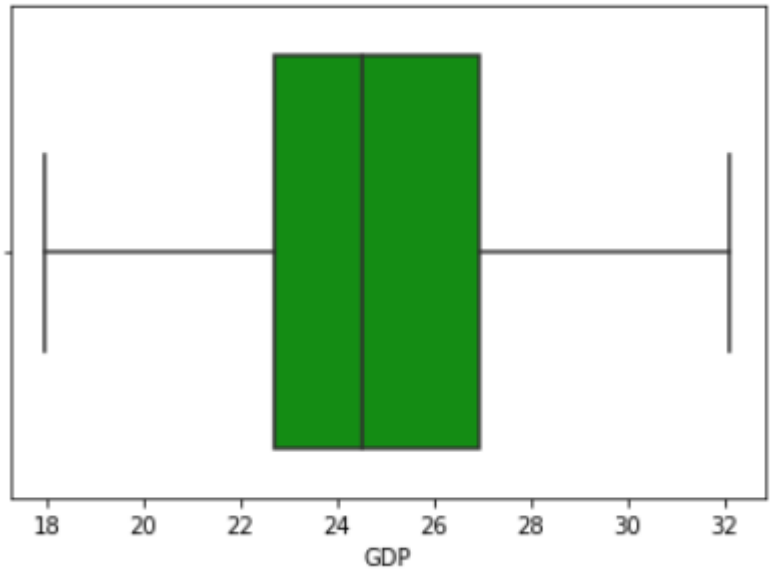

Fig 2.3.5.3 Outlier Imputation Boxplot

6. **Reg Plot**: It provides a high-level interface for drawing attractive and informative statistical graphics**.**

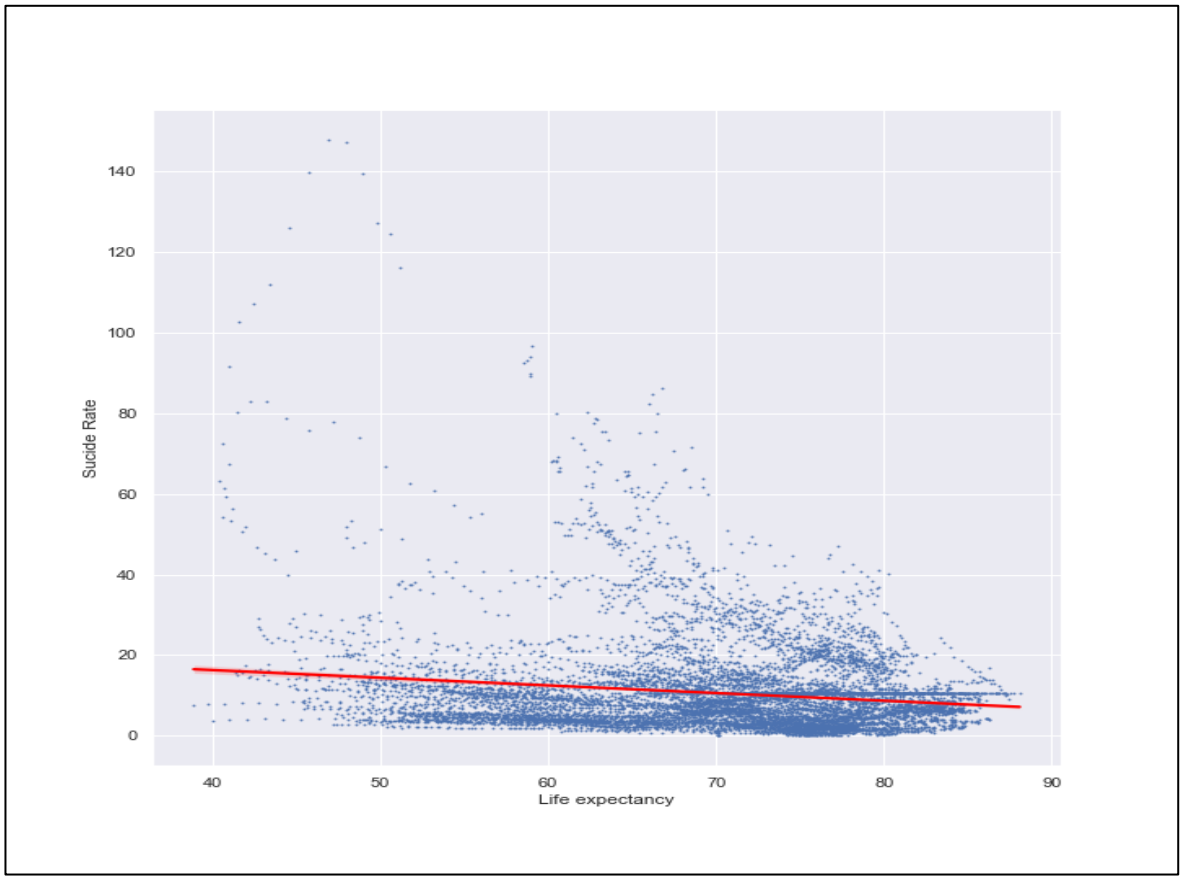

Fig 2.3.6 Reg Plot

7. **Correlation**: This is done to check the relation between different columns of dataset and with target column.

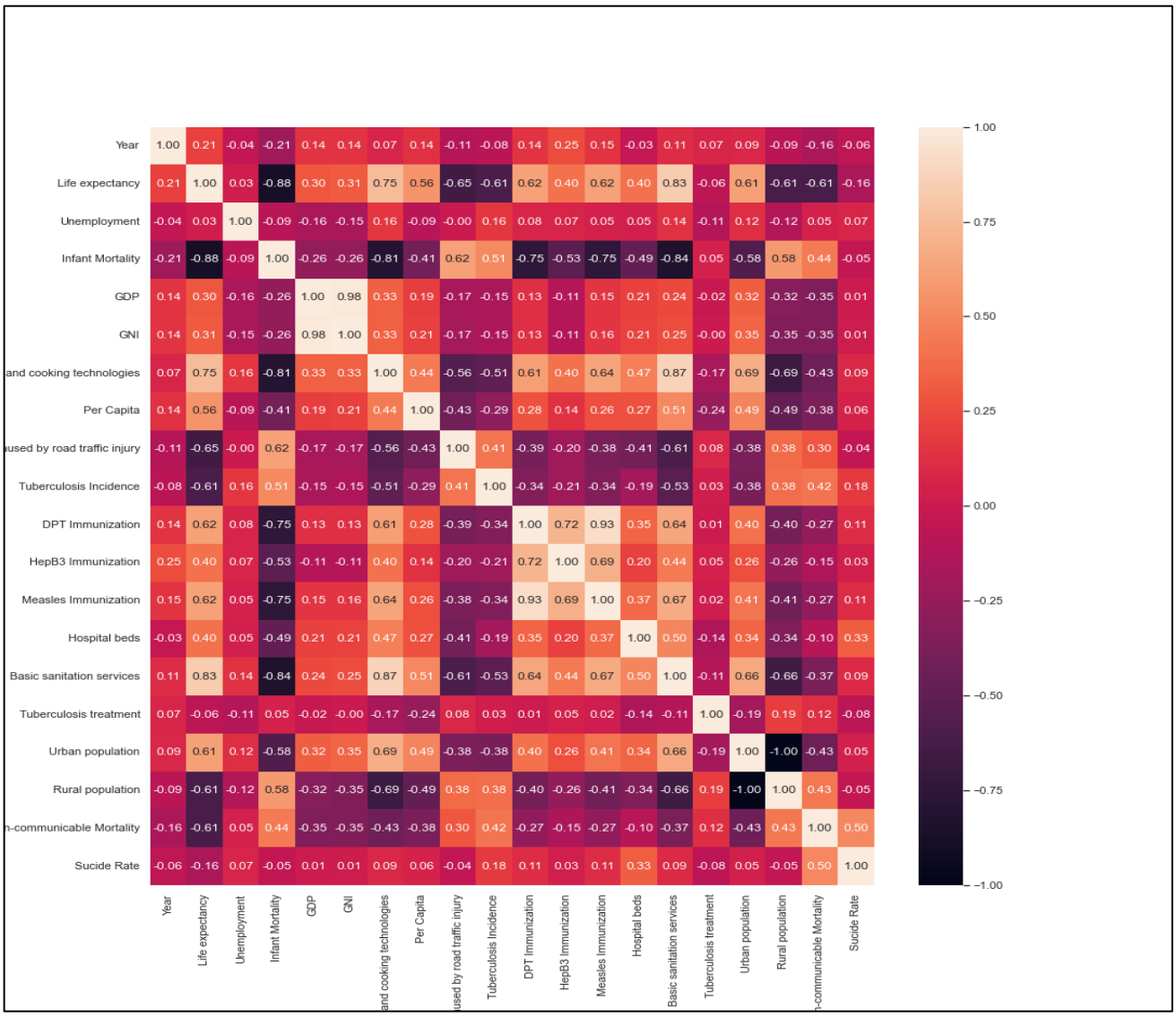

Fig 2.3.7 Heatmap

2.4 Model Building:

2.4.1 Train/Test split:

One important aspect of all machine learning models is to determine their accuracy. Now, in order to determine their accuracy, one can train the model using the given dataset and then predict the response values for the same dataset using that model and hence, find the accuracy of the model. A better option is to split our data into two parts: first one for training our machine learning model, and second one for testing our model.

In machine learning, train/test split splits the data randomly, as there's no dependence from one observation to the other. That's not the case with time series data. We have split the data sequentially. Here, you'll want to use values at the rear of the dataset for testing and everything else for training.

- Split the dataset into two pieces: a training set and a testing set.
- Train the model on the training set.
- Test the model on the testing set, and evaluate how well our model did.

Advantages of train/test split:

- Model can be trained and tested on different data than the one used for training.
- Response values are known for the test dataset, hence predictions can be evaluated
- Testing accuracy is a better estimate than training accuracy of out-of-sample performance.

Machine learning consists of algorithms that can automate analytical model building. Using algorithms that iteratively learn from data, machine learning models facilitate computers to find hidden insights from Big Data without being explicitly programmed where to look.

We have used the following three algorithms to build predictive model.

2.4.2 Standardize the data:

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data. We are using standard scalar model for this purpose which standardize features by removing the mean and scaling to unit variance.

```
# Doing the pre-processing part on training and testing set such as fitting the Standard scale.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

transformed_X_train = scaler.fit_transform(X_train)
transformed_X_test = scaler.transform(X_test)
```

Fig 2.4.2.1 Standardization

2.4.3 Feature Selection:

A feature selection method is proposed to select a subset of variables in principal component analysis (PCA) that preserves as much information present in the complete data as possible. The information is measured by means of the percentage of consensus in generalised Procrustes analysis. Principal component analysis (PCA) has been widely applied in the area of computer science. It is well known that PCA is a popular transform method and the transform result is not directly related to a sole feature component of the original sample.

```
# Applying PCA function on training
# and testing set of X component
from sklearn.decomposition import PCA
pca = PCA(n_components = 0.9)

pca_data = pca.fit(transformed_X_train)
```

Fig 2.4.3.1 PCA

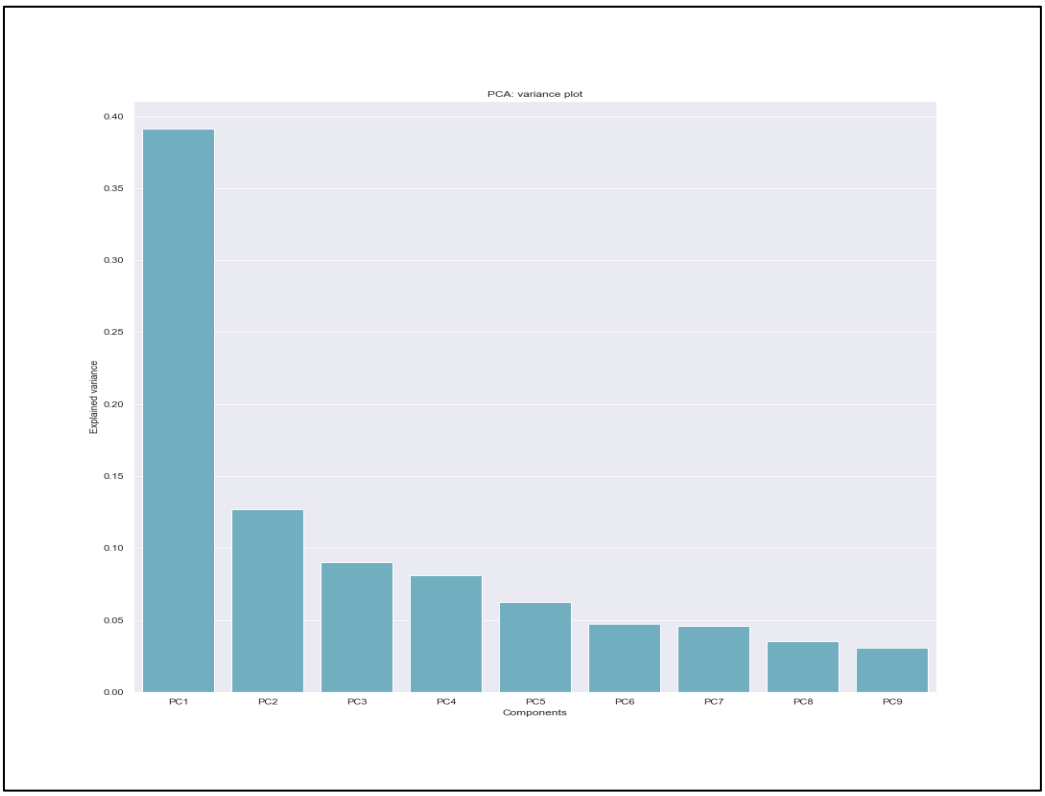

Fig 2.4.3.2 Component

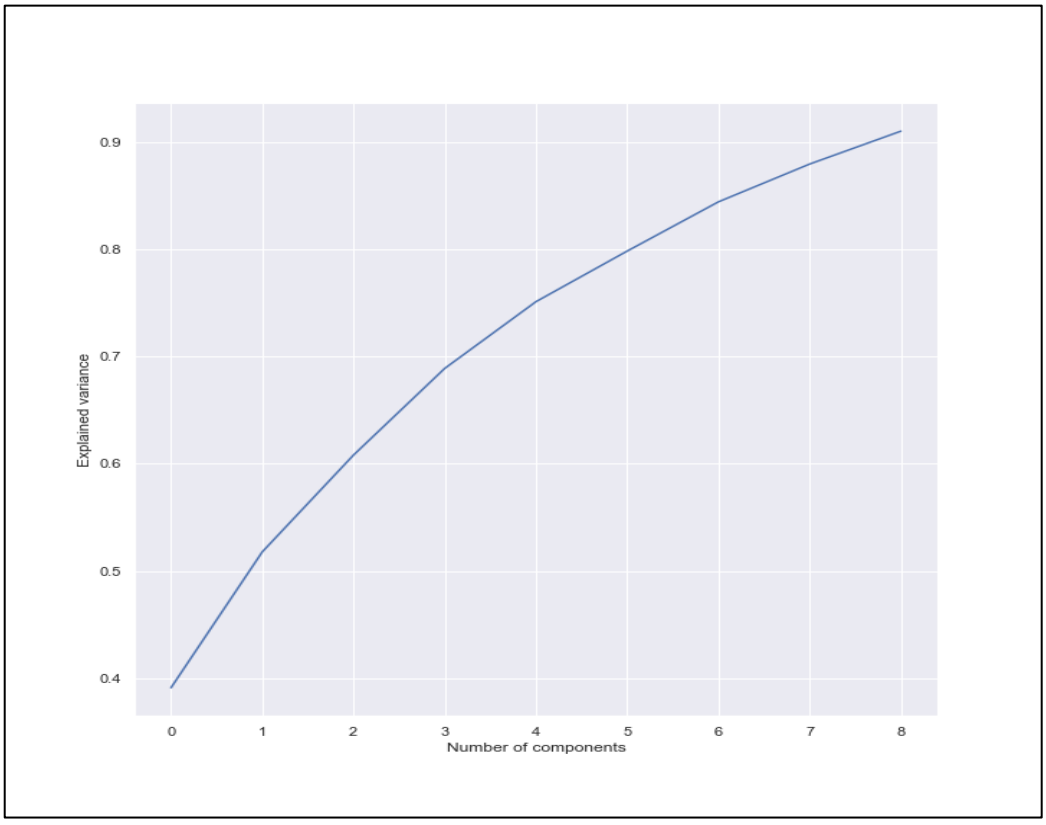

Fig 2.4.3.3 Cumulative

2.4.4 Random Forest Regressor:

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

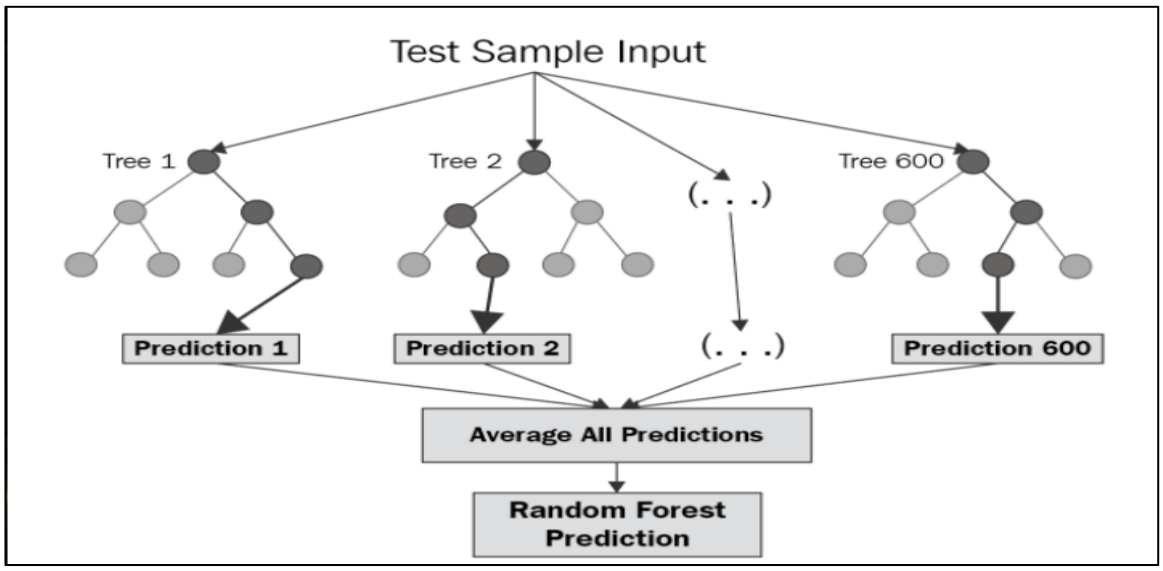

Fig 2.4.4.1 Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(oob_score = True)
```

```
from sklearn.model_selection import GridSearchCV
```

```
# param grid is dictionary
# key --> name of parameter to be tuned
# value --> list of values which you want to check
param_grid = {
    'n_estimators' : [50, 200, 500], # number of trees in the random forest
    'criterion' : ['squared_error', 'absolute_error', 'poisson'], #The function to measure the quality of a split.
    'max_depth' :[3, 5, 7], # maximum number of levels allowed in each decision tree
    'max_features' : ['sqrt', 'log2'], # number of features in consideration at every split
    'max_samples' : [0.7, 0.8, 0.9] #
}
```

```
gscv =GridSearchCV(estimator = rf, param_grid = param_grid, cv = 2, verbose =2)
```

```
gscv.fit(X_train,Y_train.ravel())
```

```
Fitting 2 folds for each of 162 candidates, totalling 324 fits
[CV] END criterion=squared_error, max_depth=3, max_features=sqrt, max_samples=0.7, n_estimators=50; total time=   0.0s
[CV] END criterion=squared_error, max_depth=3, max_features=sqrt, max_samples=0.7, n_estimators=50; total time=   0.0s
[CV] END criterion=squared_error, max_depth=3, max_features=sqrt, max_samples=0.7, n_estimators=200; total time=   0.4s
```

Fig 2.4.4.2 Applying Random Forest Regressor

Evaluation of Random Forest Regressor

```
# RF model evaluation
# calculate mean squared error (MSE) , mean absolute error (MAE) and R2 score for RF regressor.

r2, mse, mae = eval_fun(Y_test, Y_pred_rf)
print("r2 score = ", r2, "mse = ", mse, " mae = ", mae)
```

```
r2 score =  0.8967847540645739 mse =  8.909972113588067  mae =  2.2408291399797147
```

Fig 2.4.4.3 Evaluation of Random Forest Regressor

Create the model based on best params

```
# create a RF regressor
rf = RandomForestRegressor(oob_score = True, criterion = gscv_param['criterion'], max_depth = gscv_param['max_depth'],
                           max_features = gscv_param['max_features'], max_samples = gscv_param['max_samples'],
                           n_estimators = gscv_param['n_estimators'])
rf.fit(X_train,Y_train.ravel())
```

```
RandomForestRegressor(criterion='absolute_error', max_depth=7,
                      max_features='log2', max_samples=0.7, n_estimators=500,
                      oob_score=True)
```

```
Y_pred_rf = rf.predict(X_test)
```

Figure 2.4.4.4 RF with best params

```
res[['Y_test', 'Y_predicted_rf']].plot()
```

```
<AxesSubplot:>
```

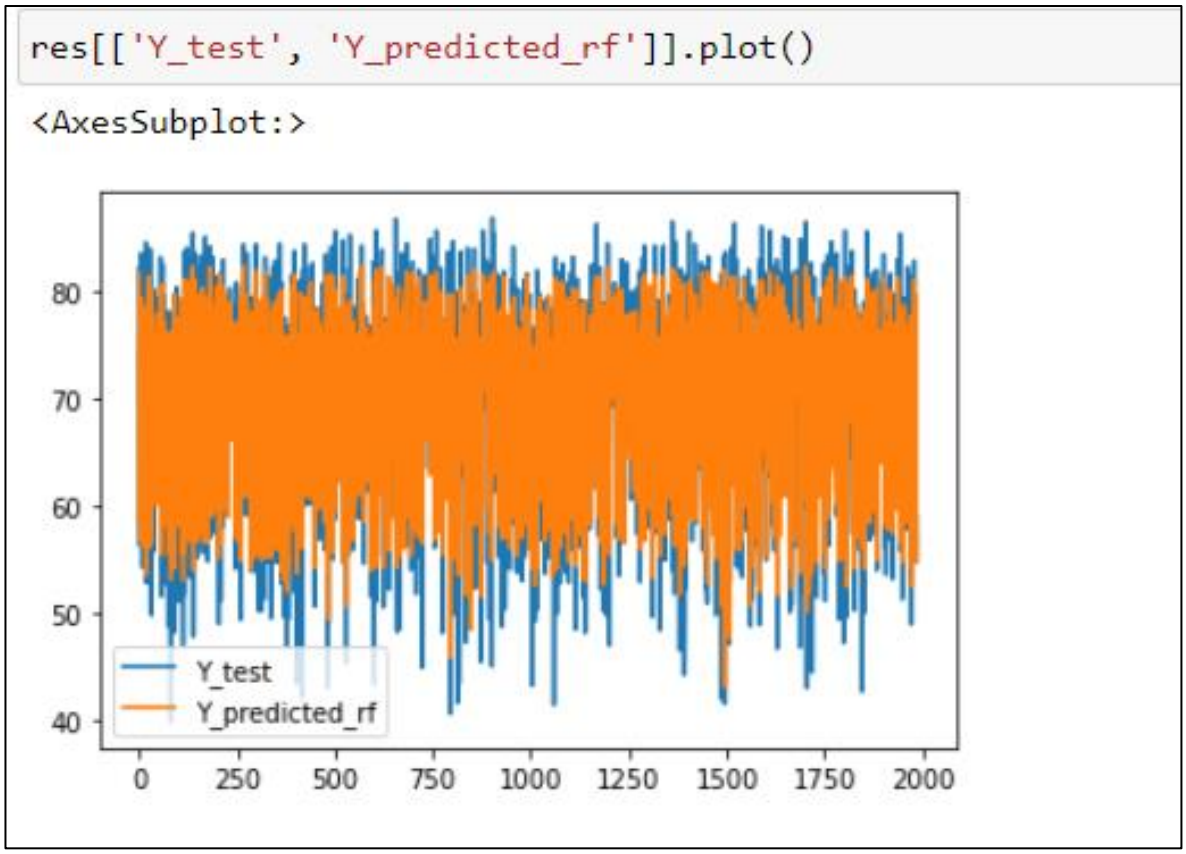

Figure 2.4.4.5 Actual and Predicted with RF

2.4.5 Gradient Boosting Regressor:

This estimator builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

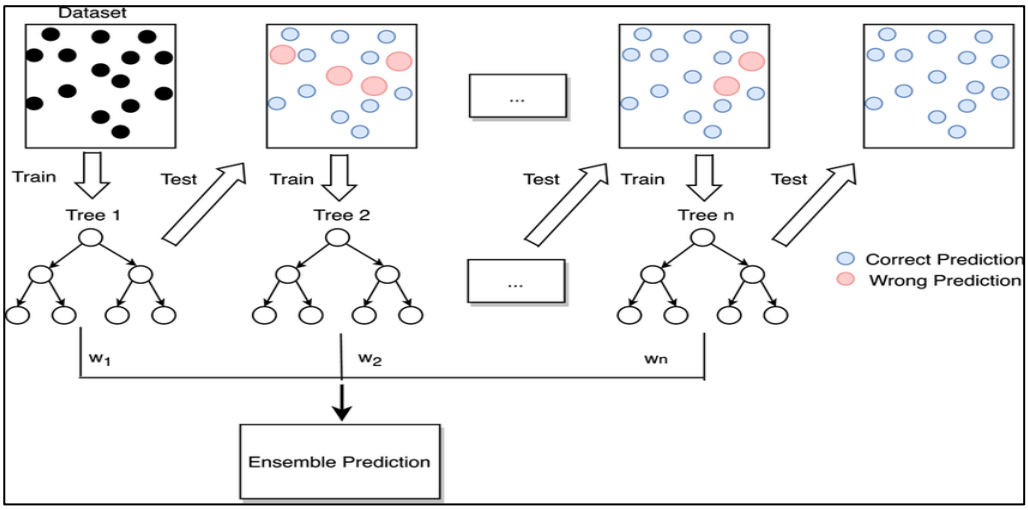

Fig 2.4.5.1 Gradient Boosting Regressor

Apply XGBoost (from SKlearn) Grid Search

```
from sklearn.ensemble import GradientBoostingRegressor
xgbc = GradientBoostingRegressor(random_state = 7)
```

```
from sklearn.model_selection import GridSearchCV
```

```
# param grid is dictionary
# key --> name of parameter to be tuned
# value --> list of values which you want to check
param_grid = {
    'learning_rate' : [0.01, 0.03],
    'n_estimators' : [50, 200, 500, 700],
    'max_depth' :[3, 5, 7, 9]
}
```

```
gscv = GridSearchCV(estimator = xgbc, param_grid = param_grid, cv = 2, verbose = 2)
```

```
gscv.fit(X_train,Y_train.ravel())
```

```
Fitting 2 folds for each of 32 candidates, totalling 64 fits
[CV] END ...learning_rate=0.01, max_depth=3, n_estimators=50; total time=   0.3s
[CV] END ...learning_rate=0.01, max_depth=3, n_estimators=50; total time=   0.3s
[CV] END ...learning_rate=0.01, max_depth=3, n_estimators=200; total time=   1.6s
```

Fig 2.4.5.2 Applying Gradient Boosting Regressor

Create the model based on best params

```
# create a GB regressor
gbr = GradientBoostingRegressor(random_state = 7, learning_rate = gscv_param['learning_rate'],
                                max_depth = gscv_param['max_depth'], n_estimators = gscv_param['n_estimators'])
gbr.fit(X_train,Y_train.ravel())
```

```
GradientBoostingRegressor(learning_rate=0.03, max_depth=7, n_estimators=700,
                          random_state=7)
```

```
Y_pred_gbr = gbr.predict(X_test)
```

Fig 2.4.5.3 XGBoost with best params

Evaluation of Gradient Boosting Regressor

```
# XG Boost model evaluation
# calculate mean squared error (MSE) , mean absolute error (MAE) and R2 score for GB regressor.

r2, mse, mae = eval_fun(Y_test,Y_pred_gbr)
print("r2 score = ", r2, "mse = ", mse, " mae = ", mae)
```

```
r2 score =  0.9668021874436133 mse =  2.8657741540875024  mae =  1.1408263426147374
```

Fig 2.4.5.4 Evaluation of Gradient Boosting Regressor

```
res[['Y_test', 'Y_predicted_gbr']].plot()
```

```
<AxesSubplot:>
```

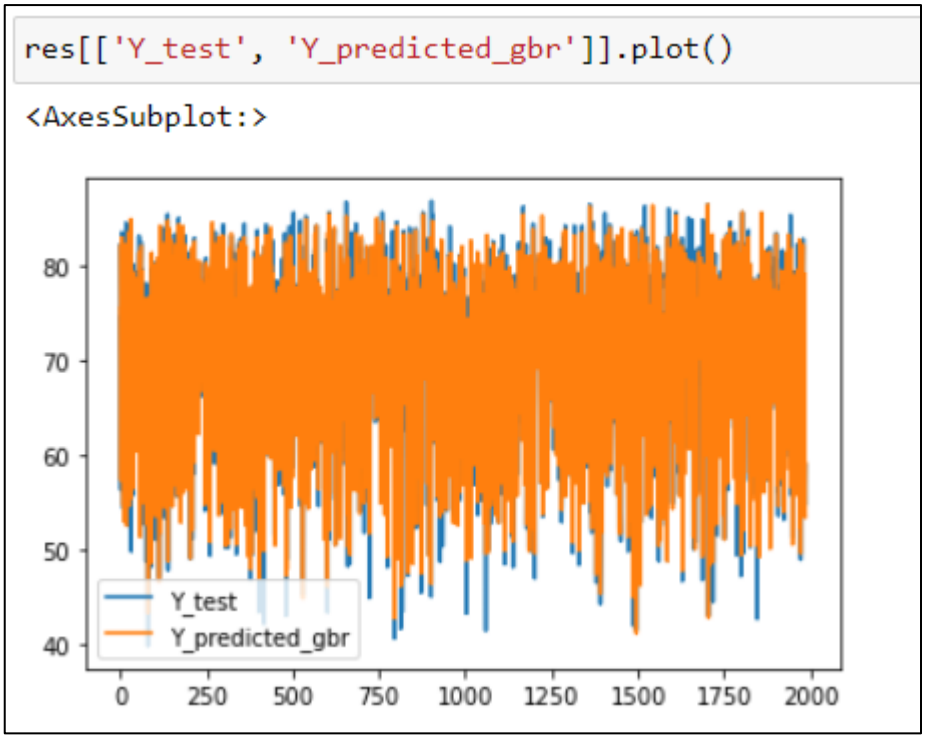

Fig 2.4.5.5 Actual and Predicted with Gradient Boosting Regressor

2.4.6 Comparison of Evaluations

Model	MAE	MSE	R2 score
Random Forest Regressor	**2.240**	**8.909**	**0.896**
Gradient Boosting Regressor	**1.140**	**2.865**	**0.966**

From the above table, we can conclude that the Gradient Boosting Regressor has minimum errors as compare to Random Forest Regressor. Gradient Boosting Regressor performs best fit.

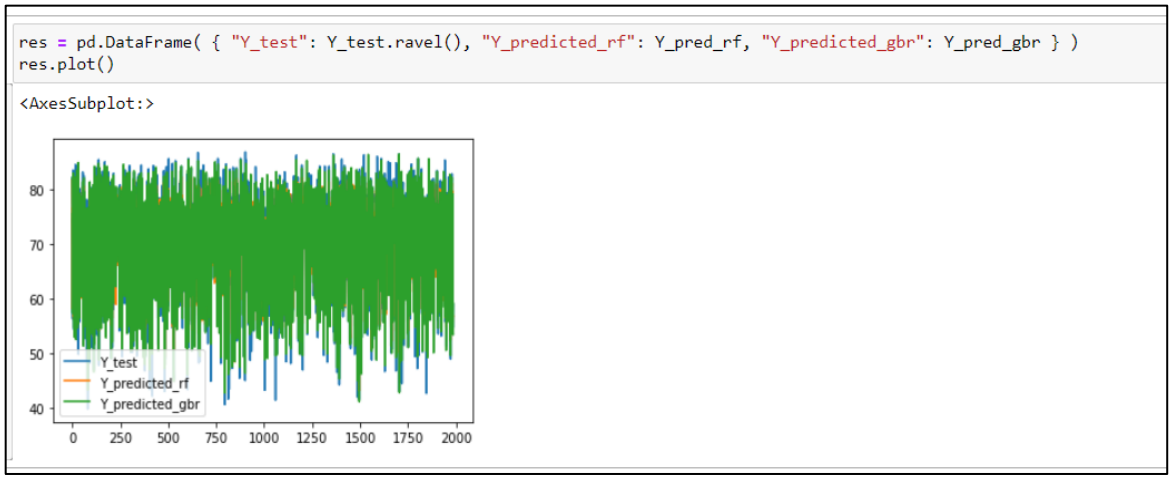

Fig 2.4.6.1 Comparison using two models

3. Requirements Specification

3.1 Hardware Requirement:

- 100 GB hard drive (Minimum requirement)
- 8 GB RAM (Minimum requirement)
- PC x64-bit CPU

3.2 Software Requirement:

- Windows/Mac/Linux
- Python- 3.9.12
- VS Code/Anaconda/Jupyter
- Libraries:
    1. Pandas
    2. Numpy
    3. Missingno
    4. Matplotlib
    5. Seaborn
    6. Sklearn
- Any Modern Web Browser like Google Chrome, Mozilla Firefox, Microsoft Edge.

4. Conclusion

- In this project, we have built models for predicting life expectancy around worldwide.
- The main aim was to analyse the impact of features on outcome and how it varies
- For model building, we gathered data from
- We performed Exploratory Data Analysis (EDA), filled missing values, resampling of the data for different time to see underlying patterns.
- Build 2 models namely Random Forest Regressor and Gradient Boosting Regressor of Bagging and Boosting respectively on clean data and evaluated data.
- After evaluation, among different models, Gradient Boosting Regressor performs best fit with an MAE = 1.140 , MSE = 2.865, R2 Score =0.966 on test set .

5. Future Scope

- This project is quite useful in studying the life structure and living patterns of population.

- Life Expectancy predictions are used in field of research and policy making decisions. Numerous gov healthcare policies can be prepared using these results.

- Generic Life Expectancy predictions can also help population to improve their lifestyles and to make efficient, healthier decisions individually.

6. References

Dataset :-

https://www.kaggle.com/datasets/kiranshahi/life-expectancy-dataset

Code References :-

https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

https://intellipaat.com/blog/what-is-random-forest-algorithm-in-python/

https://www.youtube.com/watch?v=hpT7dYsghjM&list=PLZoTAELRMXVPjaAzURB77Kz0YXxj65tYz&index=7