**DATA ANALYSIS ON WEARABLE TECHNOLOGY**

**FROM HEALTHCARE SYSTEM**

Presented by,

B.Rahul

Contents

* Introduction
* Data Representation
* Preprocessing
* Data import
* Data cleaning
* Data visualize
* Applying ML algorithms
* Logistic Regression
* KNN
* SVM
* Comparison of accuracy of LR,KNN and SVM

Explainable AI

* SHAP
* LIME
* EBM
* Features
* Execution time
* Conclusion

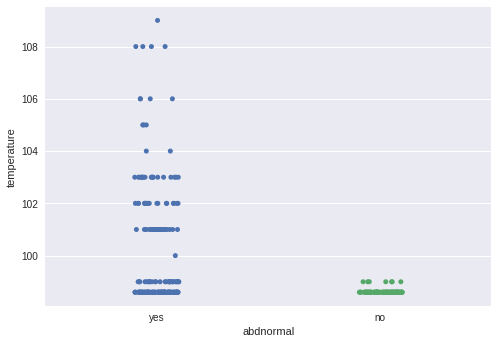
INTRODUCTION:

We are going to analyse the data from wearable devices connected with healthcare system. The actual parameter are introduced such as age,cholesterol,heart rate,blood pressure,o2,temperature as a result we will be predicting whether the patient is on normal condition by evaluating the available dataset.

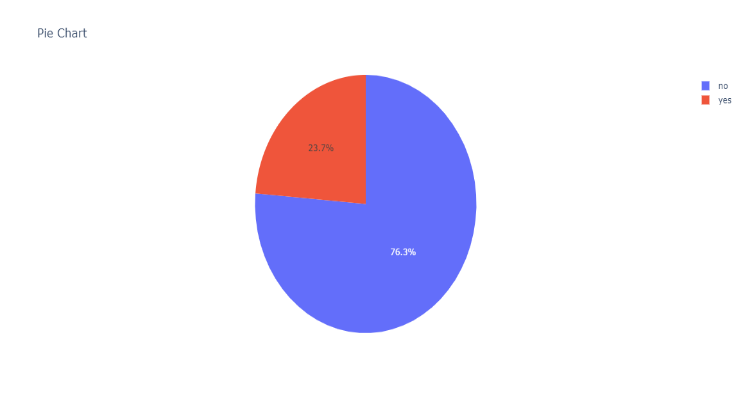
The technology has an real time application of measuring medical parameters of a patient and making a justified report of him/her health condition on certain conditions because of this feature the doctors as well as care taker of the patient can monitor the health of patient constantly.

Input data visualize:

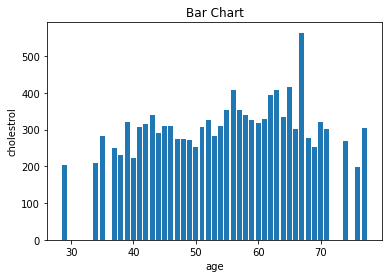
\*input for temperature and abnormal condition.



\*input for age and checking abnormal condition of the patient:



\*input for age and cholesterol level



Preprocessing:

For data import:

\*) The data is imported from the excel using read excel syntax.

Syntax: import pandas as pd

df=pd.read\_excel('health\_data.xlsx')

df.head()

Output:

| **index** | **age** | **sex** | **cp** | **trestbps** | **chol** | **fbs** | **restecg** | **thalach** | **exang** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 |

**For** **data** **cleaning**:

\*)info() method prints information about the dataframe.

Syntax: df.info()

Output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 303 entries, 0 to 302

Data columns (total 20 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 303 non-null int64

1 sex 303 non-null int64

2 cp 303 non-null int64

3 trestbps 303 non-null int64

4 chol 303 non-null int64

5 fbs 303 non-null int64

6 restecg 303 non-null int64

7 thalach 303 non-null int64

8 exang 303 non-null int64

9 oldpeak 303 non-null float64

10 slope 303 non-null int64

11 ca 303 non-null int64

12 thal 303 non-null int64

13 target 303 non-null int64

14 heart rate 303 non-null int64

15 temperature 303 non-null float64

16 blood pressure 303 non-null int64

17 o2 303 non-null int64

18 abdnormal 303 non-null object

19 Unnamed: 19 1 non-null float64

dtypes: float64(3), int64(16), object(1)

memory usage: 47.5+ KB

dtypes: float64(2), int64(14), object(3), memory usage: 45.1+ KB

\*)value\_counts()used to return series count of values.

Syntax: df.value\_counts()

Output:

age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal target heart rate temperature blood pressure o2 abdnormal Unnamed: 19

59 1 0 164 176 1 0 90 0 1.0 1 2 1 0 81 98.6 112 95 no 31.0 1

dtype: int64

\*)Dropna()method is used to drop the null values.

Syntax: df.dropna()

Output:

age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal target heart rate temperature blood pressure o2 abdnormal Unnamed: 19

297 59 1 0 164 176 1 0 90 0 1.0 1 2 1 0 81 98.6 112 95 no

\*)To remove the nan value column from excel data

Syntax: del df['Unnamed: 19']

df.isnull().any()

Output:

age False

sex False

cp False

trestbps False

chol False

fbs False

restecg False

thalach False

exang False

oldpeak False

slope False

ca False

thal False

target False

heart rate False

temperature False

blood pressure False

o2 False

abdnormal False

dtype: bool

\*)Replacing the values of ca with fbs:

Syntax: df2=df.replace({'ca':'fbs'})

print(df2)

Output:

age sex cp trestbps chol fbs restecg thalach exang oldpeak \

0 63 1 3 145 233 1 0 150 0 2.3

1 37 1 2 130 250 0 1 187 0 3.5

2 41 0 1 130 204 0 0 172 0 1.4

3 56 1 1 120 236 0 1 178 0 0.8

4 57 0 0 120 354 0 1 163 1 0.6

.. ... ... .. ... ... ... ... ... ... ...

298 57 0 0 140 241 0 1 123 1 0.2

299 45 1 3 110 264 0 1 132 0 1.2

300 68 1 0 144 193 1 1 141 0 3.4

301 57 1 0 130 131 0 1 115 1 1.2

302 57 0 1 130 236 0 0 174 0 0.0

slope ca thal target heart rate temperature blood pressure o2 \

0 0 0 1 1 104 98.6 117 94

1 0 0 2 1 130 98.6 100 97

2 2 0 2 1 88 98.6 87 97

3 2 0 2 1 126 98.6 97 92

4 2 0 2 1 78 101.0 115 97

.. ... .. ... ... ... ... ... ..

298 1 0 3 0 109 98.6 97 97

299 1 0 3 0 125 98.6 95 93

300 1 2 3 0 90 98.6 88 91

301 1 1 3 0 104 98.6 103 97

302 1 1 2 0 70 98.6 113 90

abdnormal Unnamed: 19

0 yes NaN

1 yes NaN

2 yes NaN

3 no NaN

4 yes NaN

.. ... ...

298 no NaN

299 no NaN

300 no NaN

301 no NaN

302 no NaN

[303 rows x 20 columns]

\*)To delete the column in excel data

Syntax:

del df['Unnamed: 19']

df.isnull().any()

Output:

age False

sex False

cp False

trestbps False

chol False

fbs False

restecg False

thalach False

exang False

oldpeak False

slope False

ca False

thal False

target False

heart rate False

temperature False

blood pressure False

o2 False

abdnormal False

dtype: bool

Algorithm used:

1)Logistic Regression algorithm:

* The logistic regression statistic modelling technique is used when we have a binary outcome variable.
* For example: given the parameters, will the student pass or fail? Will it rain or not? etc.
* To predict and forecast likely future outcomes with the aid of available

wearable healthcare devices dataset.

* The project contains dataset report of many sample patients listed with their ages and predicts whether patient is on normal condition or not.

Formula used:

ℓ(β)=n∑i=1[yilog(πi)+(1−yi)log(1−πi)]=n∑i=1[yiXiβ−log(1+exp(Xiβ))].

Function for logistic regression:



Syntax:

X= df.iloc[:,1:18]

Y= df.iloc[:,-1]

X= df [['temperature','age','heart rate','chol']]

y= df[['abdnormal']]

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,Y,test\_size=0.4,random\_state=100)

logreg= LogisticRegression()

logreg.fit(X\_train,y\_train)

y\_pred=logreg.predict(X\_test)

print(X\_test) #test dataset

print(y\_pred) #predicted values

Output:

temperature age heart rate chol

69 99.0 62 66 209

300 98.6 68 90 193

220 98.6 63 72 407

134 101.0 41 105 306

7 101.0 44 124 263

.. ... ... ... ...

254 98.6 59 106 273

171 98.6 48 66 229

174 98.6 60 68 206

42 98.6 45 121 208

65 102.0 35 89 183

[122 rows x 4 columns]

['yes' 'no' 'yes' 'yes' 'yes' 'yes' 'yes' 'yes' 'yes' 'no' 'yes' 'yes'

'yes' 'yes' 'no' 'no' 'yes' 'no' 'yes' 'no' 'no' 'yes' 'yes' 'no' 'yes'

'no' 'yes' 'yes' 'yes' 'yes' 'no' 'no' 'yes' 'no' 'yes' 'no' 'yes' 'yes'

'no' 'yes' 'yes' 'no' 'yes' 'yes' 'yes' 'yes' 'yes' 'yes' 'yes' 'yes'

'no' 'yes' 'yes' 'yes' 'no' 'yes' 'yes' 'yes' 'yes' 'yes' 'yes' 'yes'

'yes' 'no' 'yes' 'yes' 'yes' 'yes' 'no' 'yes' 'yes' 'yes' 'no' 'no' 'no'

'yes' 'no' 'yes' 'yes' 'yes' 'yes' 'no' 'yes' 'yes' 'yes' 'yes' 'yes'

'yes' 'yes' 'yes' 'yes' 'yes' 'no' 'yes' 'yes' 'no' 'yes' 'yes' 'yes'

'yes' 'yes' 'yes' 'yes' 'no' 'yes' 'yes' 'yes' 'yes' 'yes' 'no' 'yes'

'yes' 'yes' 'yes' 'yes' 'yes' 'no' 'yes' 'no' 'no' 'no' 'yes']

**Accuracy of the logistic algorithm:**

**Syntax:**

print('Accuracy:',metrics.accuracy\_score(y\_test, y\_pred))

print('CL Report:',metrics.classification\_report(y\_test, y\_pred, zero\_division=1))

**Output:**

Accuracy: 0.7295081967213115

CL Report: precision recall f1-score support

no 0.66 0.49 0.56 43

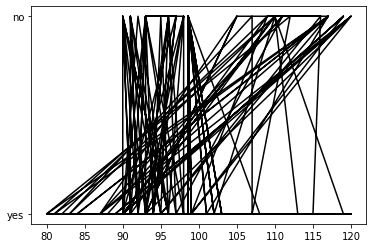
yes 0.76 0.86 0.80 79

accuracy 0.73 122

macro avg 0.71 0.67 0.68 122

weighted avg 0.72 0.73 0.72 122

Output curve:



**2)K-Nearest Neighbours algorithm:**

* K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
* It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

Formula used:

Euclidean algorithm:



**Syntax:**

#Step 2: Importing the dataset

X, y = make\_blobs(n\_samples = 500, n\_features = 2, centers = 4,cluster\_std = 1.5, random\_state = 4)

#4. Splitting Data into Training and Testing Datasets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state = 0)

#5. KNN Classifier Implementation

knn50 = KNeighborsClassifier(n\_neighbors = 50)

knn30= KNeighborsClassifier(n\_neighbors=30)

#6. Predictions for the KNN Classifiers

knn50.fit(X\_train, y\_train)

knn30.fit(X\_train, y\_train)

y\_pred\_50 = knn50.predict(X\_test)

y\_pred\_30 = knn30.predict(X\_test)

**Accuracy of the algorithm:**

**Syntax:**

from sklearn.metrics import accuracy\_score

print("Accuracy with k=50", accuracy\_score(y\_test, y\_pred\_50)\*100)

print("Accuracy with k=1", accuracy\_score(y\_test, y\_pred\_30)\*100)

from sklearn import metrics

from sklearn.metrics import classification\_report

print('Accuracy:',metrics.accuracy\_score(y\_test, pred))

print('CL Report:',metrics.classification\_report(y\_test, pred, zero\_division=1))

**Output:**

Accuracy with k=50 96.8

Accuracy with k=30 96.0

Accuracy: 0.904

CL Report: precision recall f1-score support

0 0.88 0.71 0.79 31

1 0.73 0.89 0.80 27

2 1.00 1.00 1.00 32

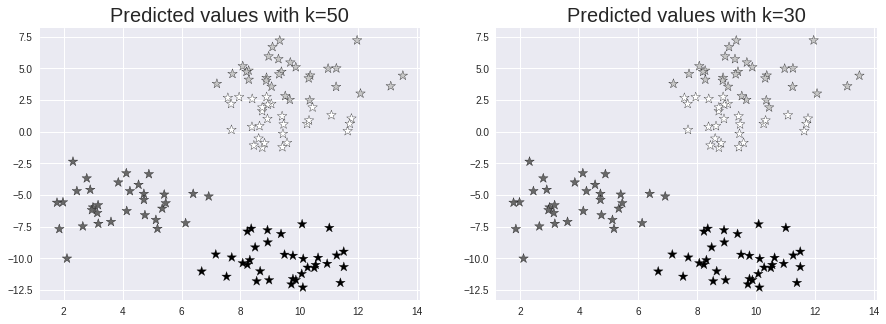
3 1.00 1.00 1.00 35

accuracy 0.90 125

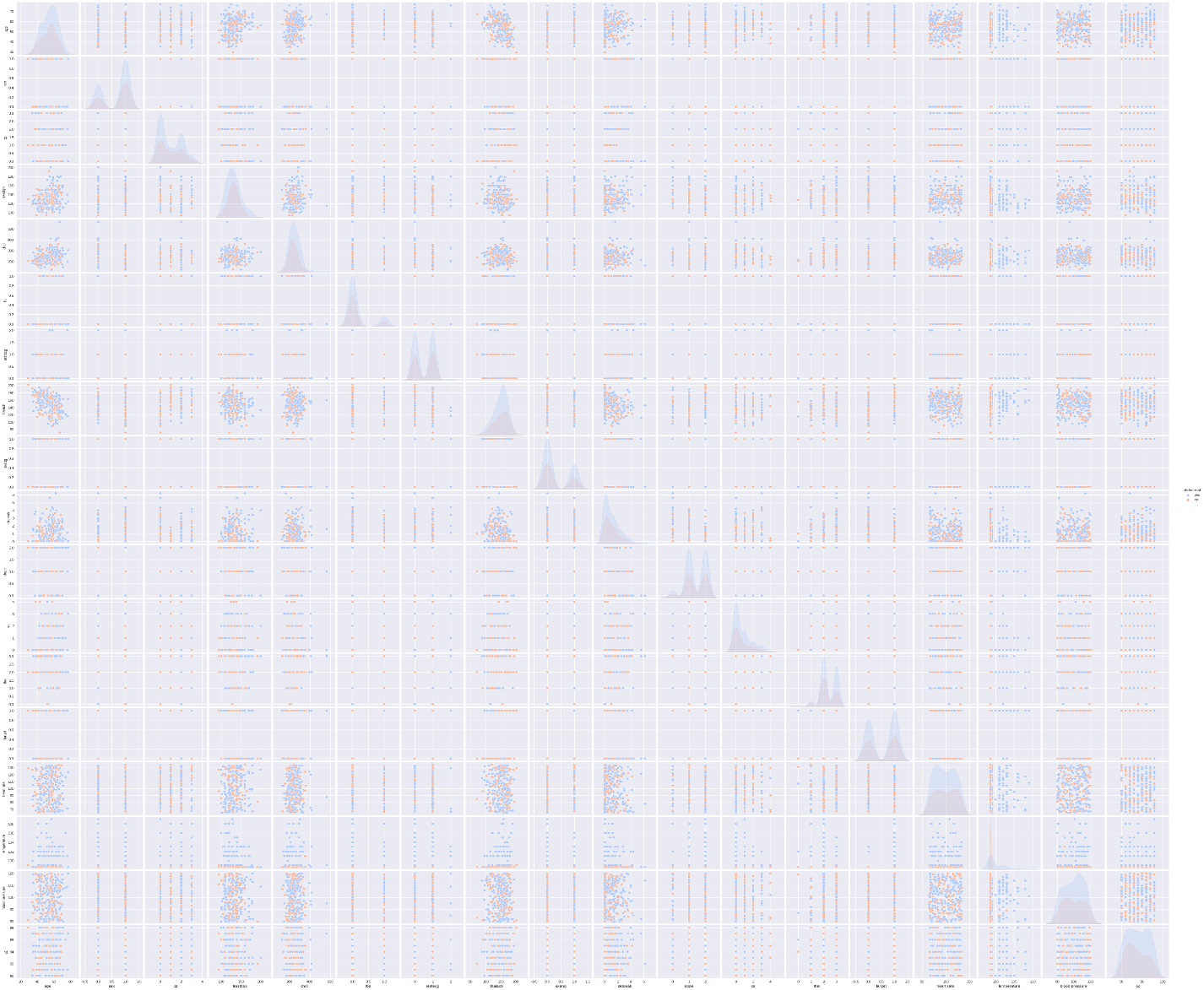
macro avg 0.90 0.90 0.90 125

weighted avg 0.91 0.90 0.90 125

**Output graph:**

****

**A plot for abnormal condition:**

****

**3)Support Vector Machine algorithm:**

* Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression.
* Though we say regression problems as well its best suited for classification.
* The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points

**Formula used: A . B = |A|** **cosθ \* |B|**

**A.B = |A| cosθ \* unit vector of B**

Where |A| cosθ is the projection of A on B and |B| is the magnitude of vector B

**Syntax:**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25)

#step 1:import data

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

#Step 2: Feature Scaling

from sklearn.svm import SVC

classifier = SVC(kernel = 'rbf', random\_state = 0)

classifier.fit(X\_train, y\_train)

#Step 3: Training the SVM Classification model on the Training Set

y\_pred = classifier.predict(X\_test)

y\_pred

#Step 4: Confusion Matrix and Accuracy

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

from sklearn.metrics import accuracy\_score

print ("Accuracy : ", accuracy\_score(y\_test, y\_pred))

print('CL Report:',metrics.classification\_report(y\_test, y\_pred, zero\_division=1))

print(cm)

Output:

Accuracy : 0.928

CL Report: precision recall f1-score support

0 0.91 0.83 0.87 35

1 0.85 0.91 0.88 32

2 0.97 1.00 0.98 29

3 1.00 1.00 1.00 29

accuracy 0.93 125

macro avg 0.93 0.93 0.93 125

weighted avg 0.93 0.93 0.93 125

[[29 5 1 0]

[ 3 29 0 0]

[ 0 0 29 0]

[ 0 0 0 29]]

#Step 5: Visualizing the Results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

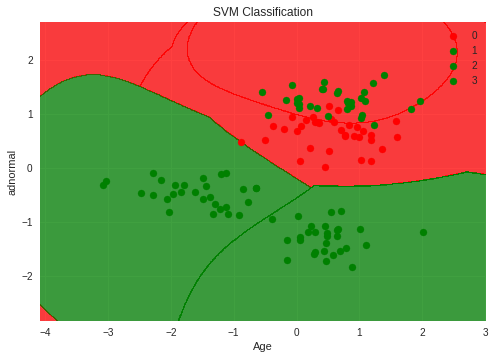
plt.title('SVM Classification')

plt.xlabel('Age')

plt.ylabel('adnormal')

plt.legend()

Output curve:



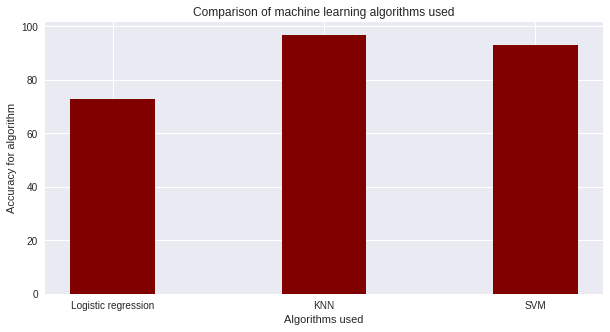
Result:

|  |  |  |  |
| --- | --- | --- | --- |
| Comparison | Logistic Regression algorithm | KNN algorithm | SVM algorithm |
| Accuracy | 73% | k=30 97% | 93% |
|  |  | k=1 96.0% |  |

(\*)KNN algorithm is best algorithm among the all other three algorithms.

Comparison plot for machine learning algorithms:

Logistic Regression ,K-Nearest Neighbour & Support Vector Machine

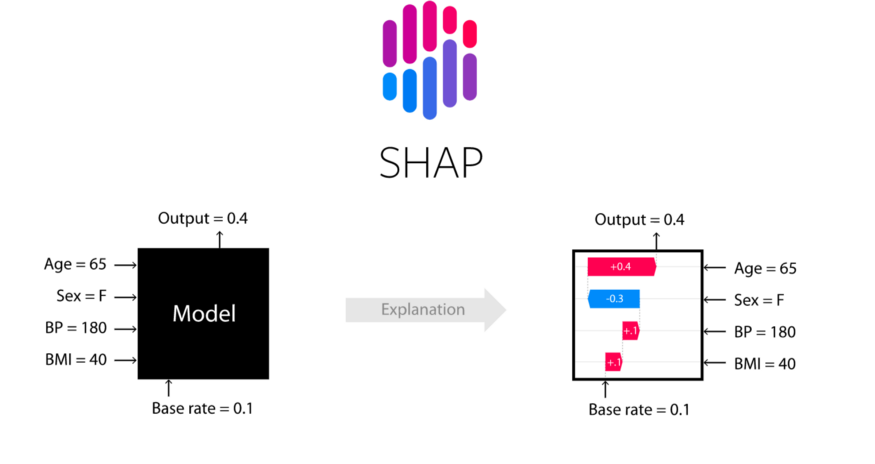


Explainable AI:

* Explainable artificial intelligence (XAI) is a set of processes and methods that allows human users to comprehend and trust the results and output created by machine learning algorithms.
* Explainable AI is used to describe an AI model, its expected impact and potential biases.
* There are seven methods used in implementation of explainable AI
* Shap
* Lime
* Shapash
* ExplainerDashboard
* Dalex
* Explainable Boosting Machine
* ELI5

1)SHAP:

* SHAP (SHapley Additive exPlanations) is a game theory based approach to explain the output of any ML model.
* It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions.



Syntax:

\*)Import these packages:

from sklearn.model\_selection import train\_test\_split

from sklearn import preprocessing

from sklearn.ensemble import RandomForestRegressor

\*)Assign values for both X and Y:

X= df[['temperature', 'o2', 'blood pressure']]

Y = df['chol']

\*)Split the dataset into two machine learning models-test and train:

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,Y,test\_size=0.4,random\_state=100)

\*)Assign an call of RandomForest regressor to model:

model = RandomForestRegressor(max\_depth=6, random\_state=0, n\_estimators=10)

\*)Implement model.fit() methos to pass both X and y values of train:

model.fit(X\_train,y\_train)

\*)Prediction values for the model created:

print(model.feature\_importances\_)

Output:

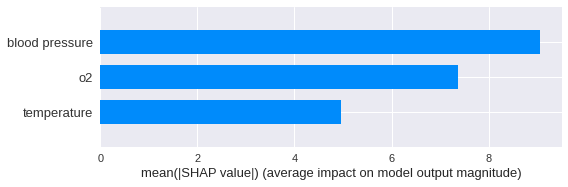
[0.20442472 0.24463035 0.55094493]

\*)Implement call of TreeExplainer model and plot shap values vs train graph model:

shap\_values=shap.TreeExplainer(model).shap\_values(X\_train)

shap.summary\_plot(shap\_values,X\_train,plot\_type="bar")

Output graph:



Syntax :

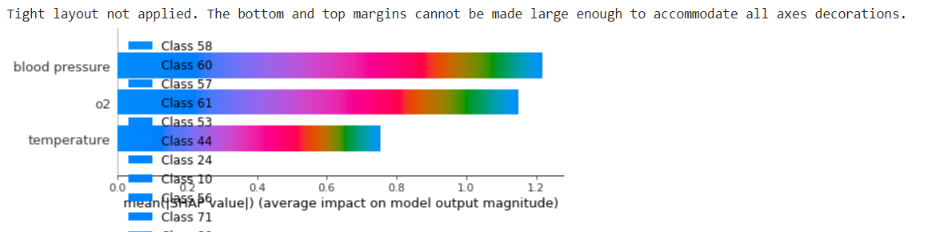
import shap

ex = shap.TreeExplainer(model)

shap\_values = ex.shap\_values(X\_test)

shap.summary\_plot(shap\_values, X\_test)

Output graph:



Syntax for effect in age factor:

fig = plt.figure(figsize=(10, 5), dpi=150,facecolor='white')

gs = fig.add\_gridspec(2, 1)

gs.update(wspace=0.11, hspace=0.5)

ax0 = fig.add\_subplot(gs[0, 0])

ax0.set\_facecolor('white')

df['age'] = df['age'].astype(int)

rate = []

for i in range(df['age'].min(), df['age'].max()):

    rate.append(df[df['age'] < i]['heart rate'].sum() / len(df[df['age'] < i]['heart rate']))

sns.lineplot(data=rate,color='#0f4c81',ax=ax0)

for s in ["top","bottom","left"]:

    ax0.spines[s].set\_visible(False)

ax0.tick\_params(axis='x', which='major', labelsize=8)

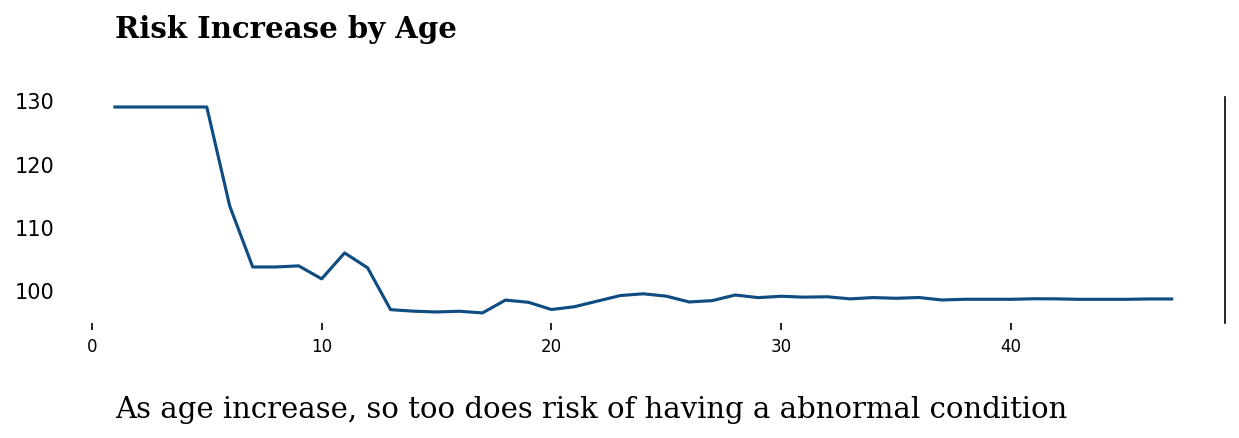
ax0.tick\_params(axis=u'y', which=u'both',length=0)

ax0.text(1,140,'Risk Increase by Age',fontsize=14,fontfamily='serif',fontweight='bold')

ax0.text(1,80,'As age increase, so too does risk of having a abnormal condition',fontsize=14,fontfamily='serif')

plt.show()

Result in increase in age factor:



Syntax for survey of people abnormal out of 20 from dataset:

from pywaffle import Waffle

fig = plt.figure(figsize=(7, 2),dpi=150,facecolor='white',

    FigureClass=Waffle,

    rows=1,

    values=[1, 19],

    colors=['#0f4c81', "lightgray"],

    characters='⬤',

    font\_size=20,vertical=True,

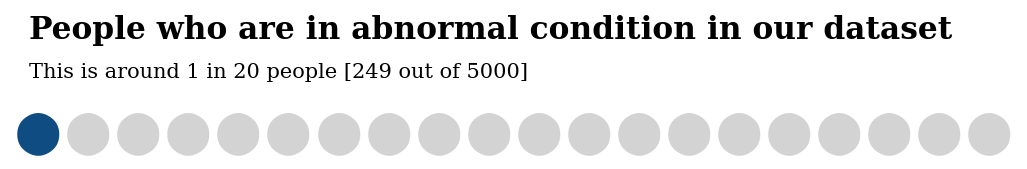
)

fig.text(0.035,0.78,'People who are in abnormal condition in our dataset',fontfamily='serif',fontsize=15,fontweight='bold')

fig.text(0.035,0.65,'This is around 1 in 20 people [249 out of 5000]',fontfamily='serif',fontsize=10)

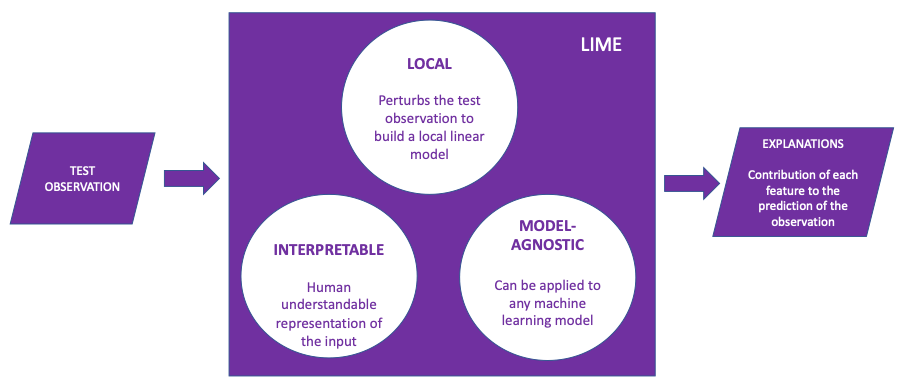
plt.show()

Result of number of people out of 20 in our dataset:



2)LIME:

* **LIME(or Local  Interpretable Model-agnostic Explanations)**
* The beauty of LIME its accessibility and simplicity.
* The core idea behind LIME though exhaustive is really intuitive and simple! Let’s dive in and see what the name itself represents:
  + **Model agnosticism** refers to the property of LIME using which it can give explanations for any given supervised learning model by treating as a ‘black-box’ separately. This means that LIME can handle almost any model that exists out there in the wild!
  + **Local explanations** mean that LIME gives explanations that are locally faithful within the surroundings or vicinity of the observation/sample being explained.
* Though LIME limits itself to supervised Machine Learning and Deep Learning models in its current state, it is one of the most popular and used XAI methods out there.



Formulae used:

We will measure unfaithfulness with the letter L.

L(f, g, ∏x) will measure how unfaithful g is when making approximations of f in the locality we defined as ∏x.

We must minimize unfaithfulness with L(f, g, ∏x) and find how to keep the complexity Ω(g) as low as possible.

Finally, we can define an explanation E generated by LIME as follows:

E(x) = L(f, g, ∏x) + Ω(g)

LIME will draw samples weighted by ∏x to optimize the equation to produce the best interpretation and explanations E(x) regardless of the model implemented.

LIME can be applied to various models, fidelity functions, and complexity measures. However, LIME’s approach will follow the method we have defined.

Syntax:

# Importing the necessary libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Loading the dataset using sklearn

from sklearn.datasets import load\_boston

data = load\_boston()

# Displaying relevant information about the data

print(data['DESCR'][200:1420])

# Separating data into feature variable X and target variable y respectively

from sklearn.model\_selection import train\_test\_split

data=pd.read\_excel('health\_data.xlsx')

X = data.iloc[:,1:18].values

y = data.iloc[:, 2].values

# Extracting the names of the features from data

features = data['abdnormal']

# Splitting X & y into training and testing set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

  X, y, train\_size=0.90, random\_state=50)

# Creating a dataframe of the data, for a visual check

df = pd.concat([pd.DataFrame(X), pd.DataFrame(y)], axis=1)

#df.columns = np.concatenate((features, np.array(['o2'])))

print("Shape of data =", df.shape)

# Printing the top 5 rows of the dataframe

df.head()

Output:

Shape of data = (303, 18)

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **0** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.0 | 3.0 | 145.0 | 233.0 | 1.0 | 0.0 | 150.0 | 0.0 | 2.3 | 0.0 | 0.0 | 1.0 | 1.0 | 132.0 | 98.6 | 116.0 | 94.0 | 3 |
| **1** | 1.0 | 2.0 | 130.0 | 250.0 | 0.0 | 1.0 | 187.0 | 0.0 | 3.5 | 0.0 | 0.0 | 2.0 | 1.0 | 114.0 | 98.6 | 90.0 | 94.0 | 2 |
| **2** | 0.0 | 1.0 | 130.0 | 204.0 | 0.0 | 0.0 | 172.0 | 0.0 | 1.4 | 2.0 | 0.0 | 2.0 | 1.0 | 65.0 | 98.6 | 102.0 | 97.0 | 1 |
| **3** | 1.0 | 1.0 | 120.0 | 236.0 | 0.0 | 1.0 | 178.0 | 0.0 | 0.8 | 2.0 | 0.0 | 2.0 | 1.0 | 108.0 | 98.6 | 92.0 | 93.0 | 1 |
| **4** | 0.0 | 0.0 | 120.0 | 354.0 | 0.0 | 1.0 | 163.0 | 1.0 | 0.6 | 2.0 | 0.0 | 2.0 | 1.0 | 124.0 | 101.0 | 85.0 | 91.0 | 0 |

# Instantiating the prediction model - an extra-trees regressor

from sklearn.ensemble import ExtraTreesRegressor

reg = ExtraTreesRegressor(random\_state=50)

# Fitting the predictino model onto the training set

reg.fit(X\_train, y\_train)

# Checking the model's performance on the test set

print('R2 score for the model on test set =', reg.score(X\_test, y\_test))

Accuracy: R2 score for the model on test set = 1.0

# Importing the module for LimeTabularExplainer

import lime.lime\_tabular

# Instantiating the explainer object by passing in the training set, and the extracted features

explainer\_lime = lime.lime\_tabular.LimeTabularExplainer(X\_train,feature\_names=features,verbose=True, mode='regression')

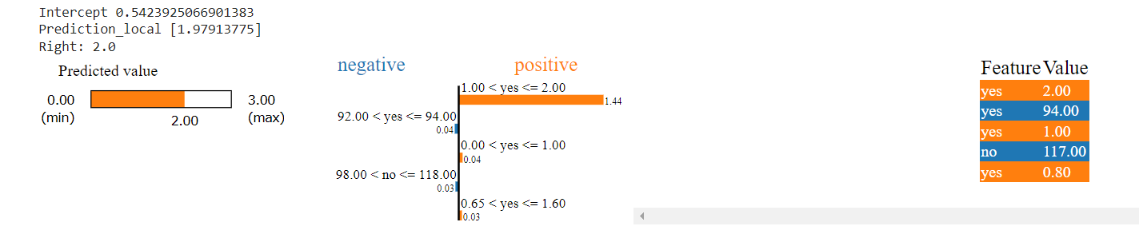
# Importing the module for LimeTabularExplainer

import lime.lime\_tabular

# Instantiating the explainer object by passing in the training set, and the extracted features

explainer\_lime = lime.lime\_tabular.LimeTabularExplainer(X\_train,feature\_names=features,verbose=True, mode='regression')

Output plot:



Syntax:

# Index corresponding to the test vector

i = 18

# Number denoting the top features

k = 5

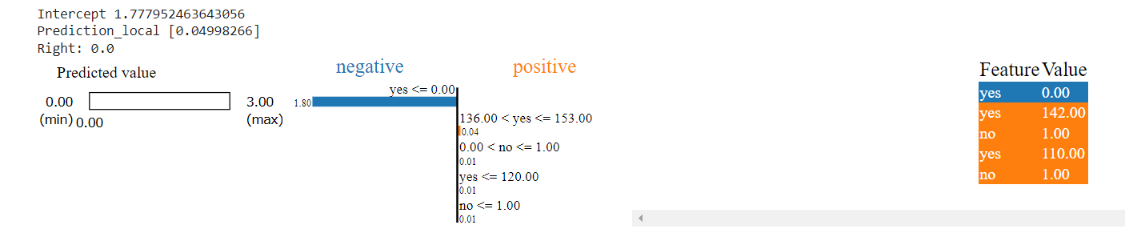
exp\_lime = explainer\_lime.explain\_instance(

  X\_test[i], reg.predict, num\_features=k)

# Finally visualizing the explanations

exp\_lime.show\_in\_notebook()

Output plot:



Syntax:

exp = explainer.explain\_instance(data\_row=X\_test.iloc[1],predict\_fn=model.predict\_proba)

exp.show\_in\_notebook(show\_table=True)

exp = explainer.explain\_instance(data\_row=X\_test.iloc[4],predict\_fn=model.predict\_proba)

exp.show\_in\_notebook(show\_table=True)

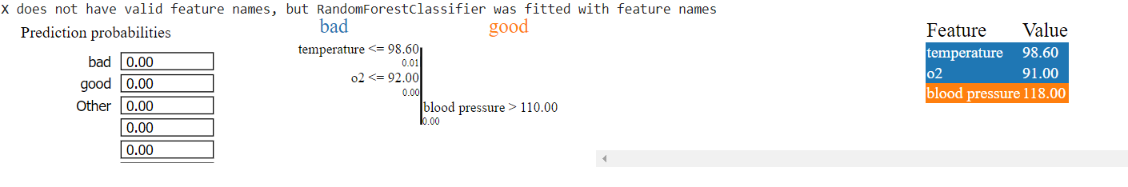
exp = explainer.explain\_instance(data\_row=X\_test.iloc[18],predict\_fn=model.predict\_proba)

exp.show\_in\_notebook(show\_table=True)

Output plot:







3)Explainable Boosting Machine:

* Explainable Boosting Machine (EBM) is a tree-based, cyclic gradient boosting Generalized Additive Model with automatic interaction detection.
* EBMs are often as accurate as state-of-the-art blackbox models while remaining completely interpretable.
* Although EBMs are often slower to train than other modern algorithms, EBMs are extremely compact and fast at prediction time.

Formulae used:

EBM is a glassbox model, designed to have accuracy comparable to state-of-the-art machine learning methods like Random Forest and Boosted Trees, while being highly intelligibile and explainable. EBM is a generalized additive model (GAM) of the form:

*g*(*E*[*y*])=*β*0​+∑*fj*​(*xj*​)

Syntax:

#ExplainableBoostingRegressor

from interpret.glassbox import ExplainableBoostingRegressor

from sklearn.datasets import load\_boston

X, y = load\_boston(return\_X\_y=True)

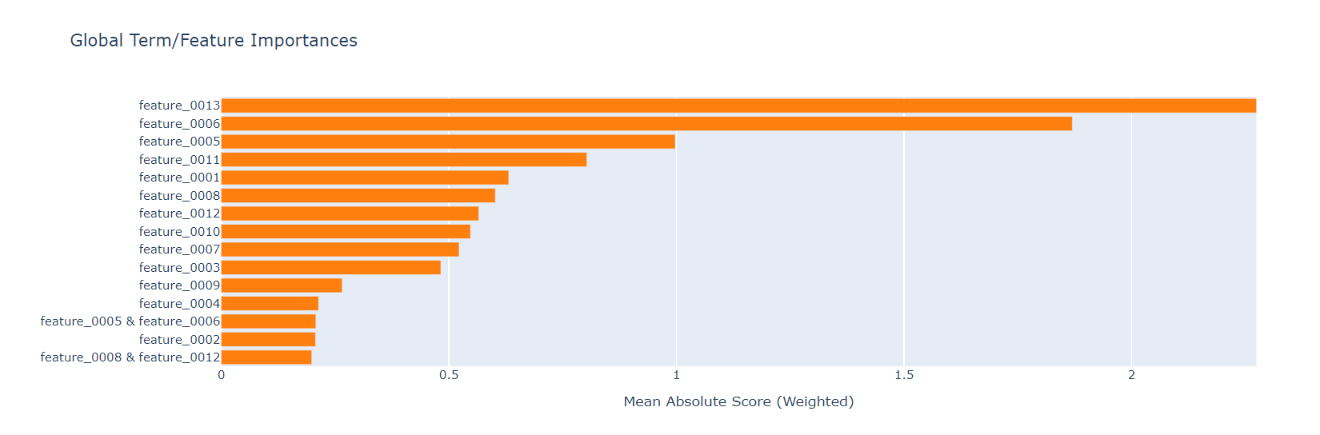
ebm = ExplainableBoostingRegressor()

ebm.fit(X, y)

from interpret import show

show(ebm.explain\_global())

Output plot:



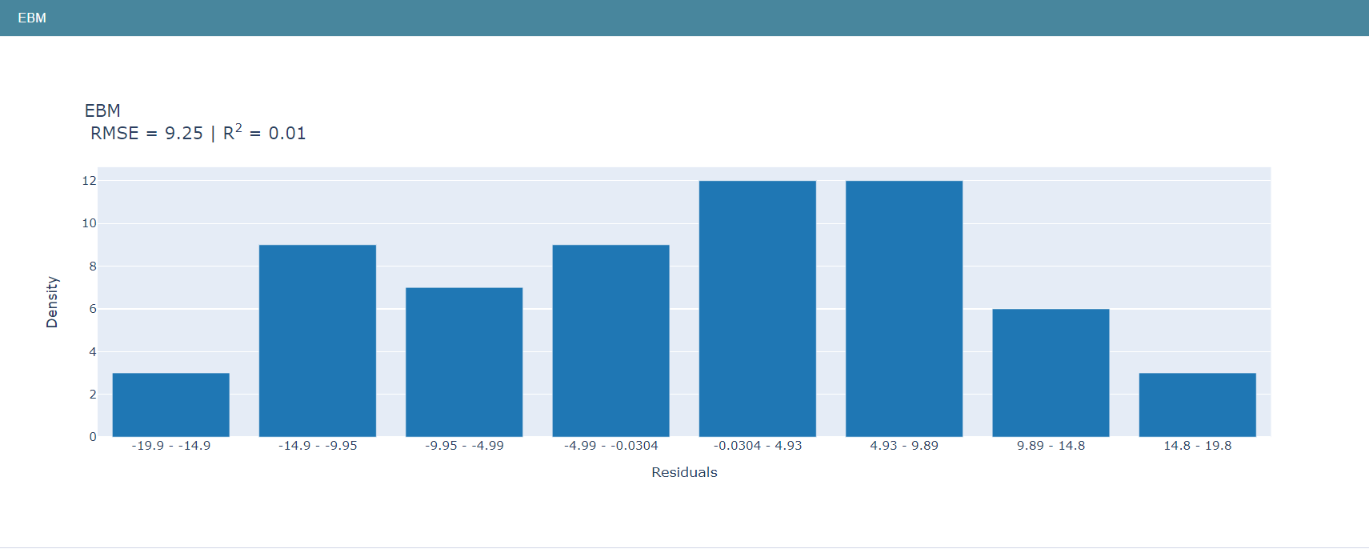
**Syntax for evalvating EBM performance:**

from interpret.perf import RegressionPerf

ebm\_perf = RegressionPerf(ebm.predict).explain\_perf(X\_test, y\_test, name='EBM')

show(ebm\_perf)

Output plot:



**Syntax for comparing EGB performance with other regressors:**

lr = LinearRegression(random\_state=seed)

lr.fit(X\_train, y\_train)

rt = RegressionTree(random\_state=seed)

rt.fit(X\_train, y\_train)

rf = RandomForestRegressor(n\_estimators=100, n\_jobs=-1)

rf.fit(X\_train, y\_train)

lr\_perf = RegressionPerf(lr.predict).explain\_perf(X\_test, y\_test, name='Linear Regression')

rt\_perf = RegressionPerf(rt.predict).explain\_perf(X\_test, y\_test, name='Regression Tree')

rf\_perf = RegressionPerf(rf.predict).explain\_perf(X\_test, y\_test, name='Blackbox')

lr\_global = lr.explain\_global(name='Linear Regression')

rt\_global = rt.explain\_global(name='Regression Tree')

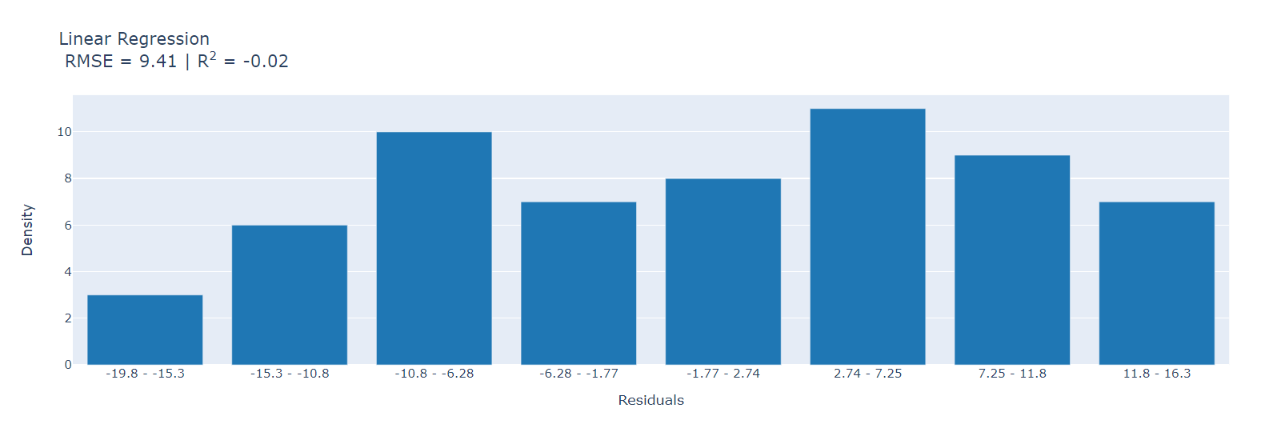
show(lr\_perf)

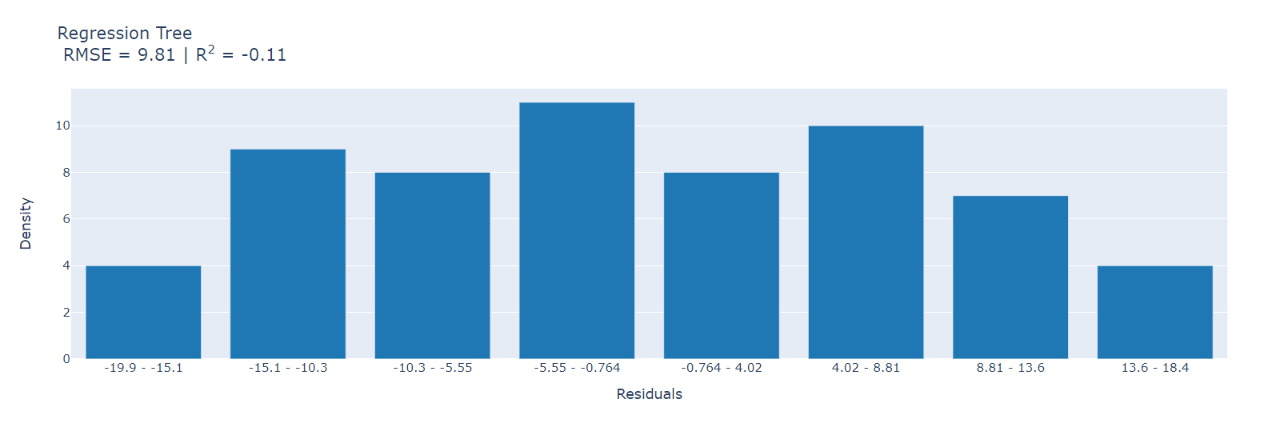
show(rt\_perf)

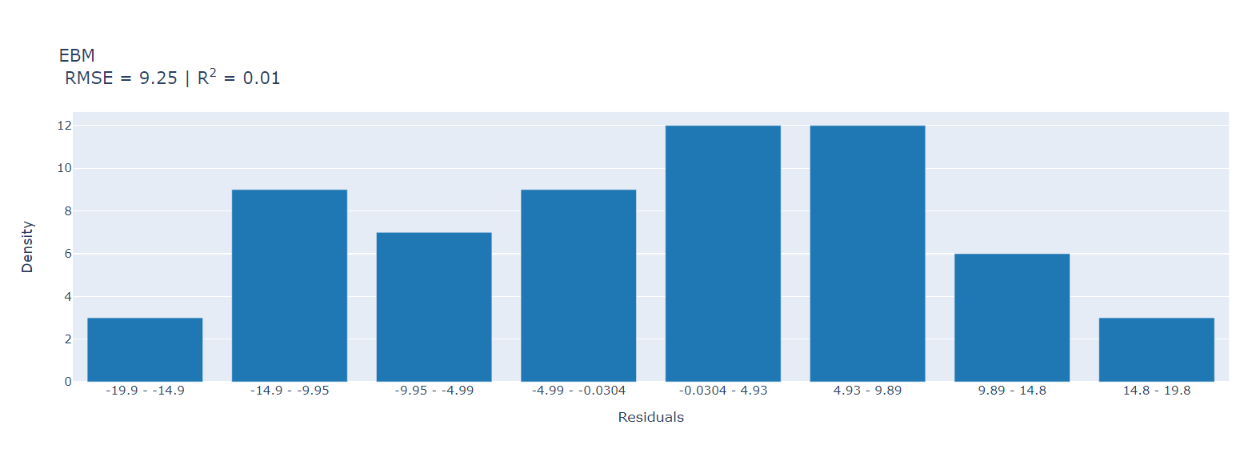
show(ebm\_perf)

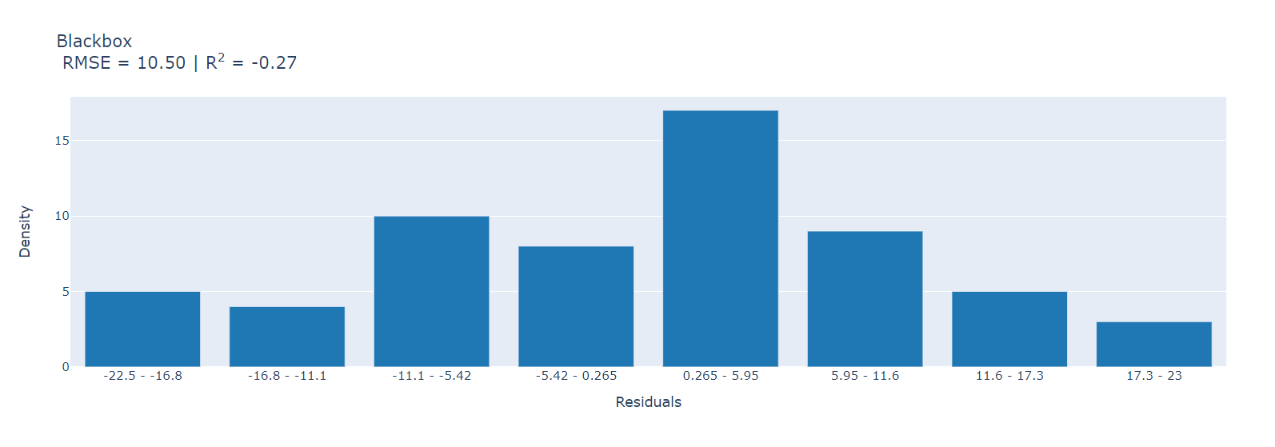
show(rf\_perf)

Output plot:









Execution time:

* The time taken to execute the entire program.

Syntax:

# Timing Python Scripts

# Import relevant modules

import time

from datetime import datetime

# Start of the python script

script\_start\_time = datetime.now()

# Extra coding...

for i in range(10000000):

    j = 10 + i

# Timing a section of python code

# This gives the start time

start\_time = time.time()

# Do some code

for i in range(10000000):

    j = 10 + i

# End time once the code has finished

end\_time = time.time()

# Total time

final\_time = end\_time - start\_time

print(final\_time)

# Script end time

script\_end\_time = datetime.now()

print(script\_end\_time - script\_start\_time)

Output time:

2.1743342876434326

0:00:04.265623

Conclusion:

In this paper, we are measuring and analysing the sample data discharged from wearable devices for health care system. The data acquire parameters like age, cholesterol level, blood pressure, heart rate, oxygen level of the patient by training and testing the prediction of normal condition of patient was founded. The supervised algorithm and unsupervised algorithm like logistic regression,knn,svm are implemented and accuracy are successfully probed for these algorithm. Best accuracy was found in knn algorithm with 94%.

Various explainable ai algorithm like shap,lime and explainable boosting machine are utilized. The performance of those algorithms and execution time of code are determined with plot.

THANK YOU !!!