HEART DISEASE PREDICTION

PROJECT 3

By:
Priyanka
Chandramohan

1. Introduction

One of the most common & terrible disease is Heart Disease now a days. What are the reasons behind it, how many people are affecting by heart disease, how people can prevent this disease, people are less aware from all these aspects. So for that understanding we require some analysis and visualization so people can understand which types of trends have followed in the past years with this factors, new generations will get benefited from this analysis & aware how they prevent it.

- **Problem Statement**

  Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide. Four out of 5CVD deaths are due to heart attacks and strokes, and one-third of these deaths occur prematurely in people under 70 years of age. Heart failure is a common event caused by CVDs and this dataset contains 11 features that can be used to predict a possible heart disease.

- **Objective**

  People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need early detection and management where in a machine learning model can be of great help.

About Dataset:

The dataset we've consisted of records of Heart Disease. It has various columns which describe the factors related with the heart disease such as the age, Chest Pain Type, Resting BP, Cholesterol, Max HR, Exercise Angina etc.
Such a dataset is very useful for the people who got affected & who don't as well, who don't affected by heart disease they can also predict & get aware with the help of MachineLearning models.

This is what the dataset looks like,

```
#reading first five rows from dataset
df.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

Table 1 - Dataset

It has lots of rows and columns, which is 918 rows spread across 12 columns.

Table 2 - Columns in

```
df.shape
```

```
(918, 12)
```

```
df.columns
```

```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

2. EDA

EDA stands for exploratory data analysis where we explore our data and grab insights from it. EDA helps us in getting knowledge in form of various plots and diagrams wherewe can easily understand the data and its features.

Analysis of the Data

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.510893	9.432617	28.0	47.00	54.0	60.0	77.0
RestingBP	918.0	132.396514	18.514154	0.0	120.00	130.0	140.0	200.0
Cholesterol	918.0	198.799564	109.384145	0.0	173.25	223.0	267.0	603.0
FastingBS	918.0	0.233115	0.423046	0.0	0.00	0.0	0.0	1.0
MaxHR	918.0	136.809368	25.460334	60.0	120.00	138.0	156.0	202.0
Oldpeak	918.0	0.887364	1.066570	-2.6	0.00	0.6	1.5	6.2
HeartDisease	918.0	0.553377	0.497414	0.0	0.00	1.0	1.0	1.0

Table 3- Checking of the Skewness

Observation:
'Age' ranges from 28 – 77. As Mean (53.510893) < Median (54.0) data is slightly left skewed or negatively skewed.

'**Resting BP'** ranges from 0 – 200. As Mean (132.396514) > Median (130.0) data is slightly right skewed or positively skewed.

'**Cholesterol'** maximum value is 603.0. As Mean (198.799564) < Median (223.0) data is left skewed or negatively skewed.

'**Fasting BS'** has Mean (0.233115) = Median (0.0) we can say it has zero skewness.

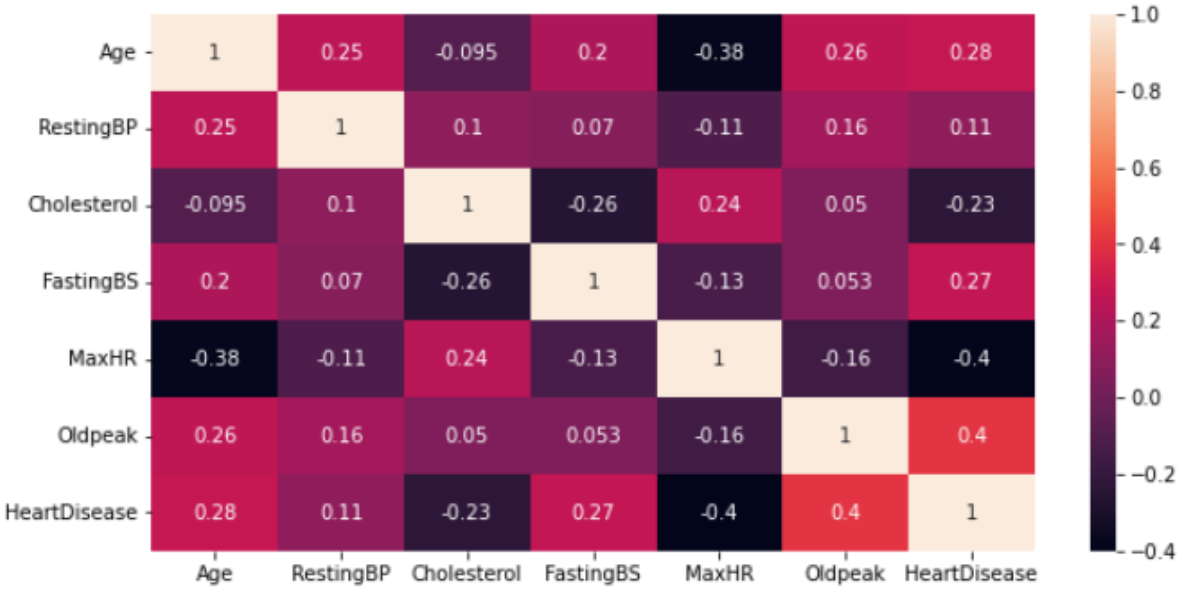
'**Max HR'** ranges from 60.0 – 202.0. As Mean (136.809368) < Median (138.0) data is slightly left skewed or negatively skewed.

'**Old peak'** ranges from -2.6 to 6.2. As Mean (0.887364) > Median (0.6) data is slightly right skewed or positively skewed.
'**HeartDisease'** has Mean (0.553377) < Median (1.0) as median is almost double than mean, we can say that data is highly left skewed or negatively skewed.

Bivariate Analysis

Heatmap

Here we are looking which feature can be dropped according to correlation betweenthem. On left the name of the column is written and on right the columns which are highly correlated with it is written.



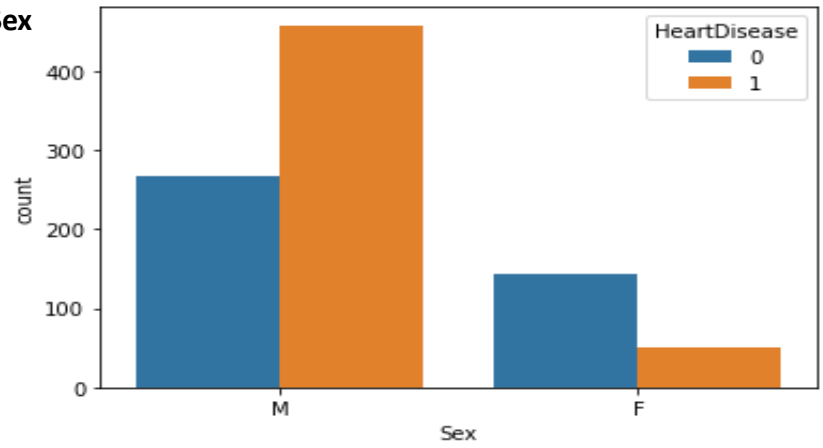
Plot 1 - Heatmap

We can see there have some positive & negative correlation, for example

- As age increases, heart disease also increases so there have a positive correlation between them
- There have a negative correlation between Fasting Blood Sugar & Cholesterol, maybe both the parameters are not affecting each other directly such as if Fasting Blood Sugar going higher, Cholesterol may goes down or normal

Bivariate Analysis of Each Column with Heart disease

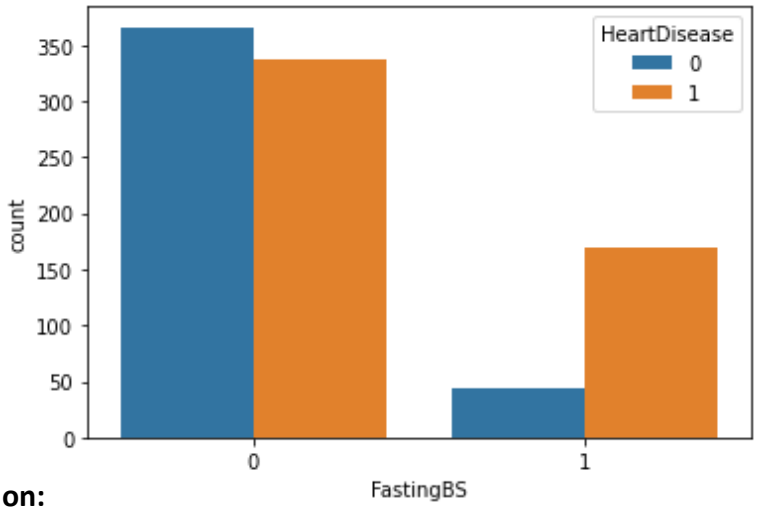
Bivariate Analysis of Sex



Observation:

Using count plot we can say males are more affected than females by heart disease

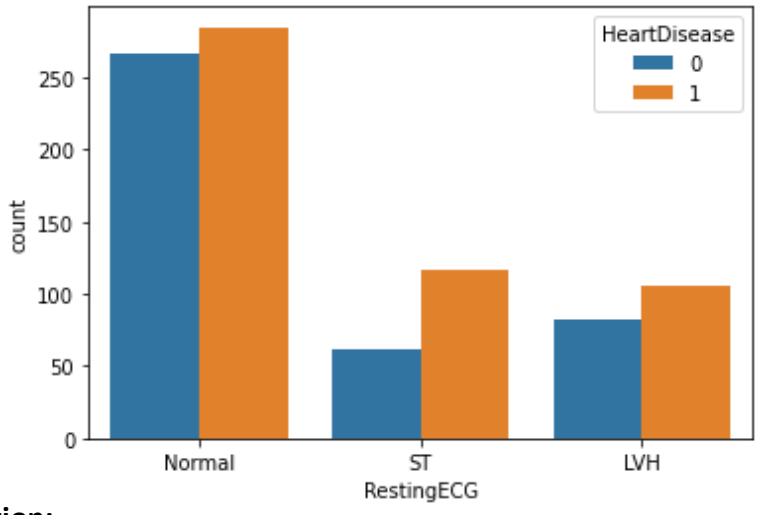
Bivariate Analysis of Fasting BS



Observation:

According to our dataset who has Fasting BS they are lesser affected by the heart disease, and who has not Fasting BS they affected in high amounts by heart disease.

Bivariate Analysis of Resting ECG



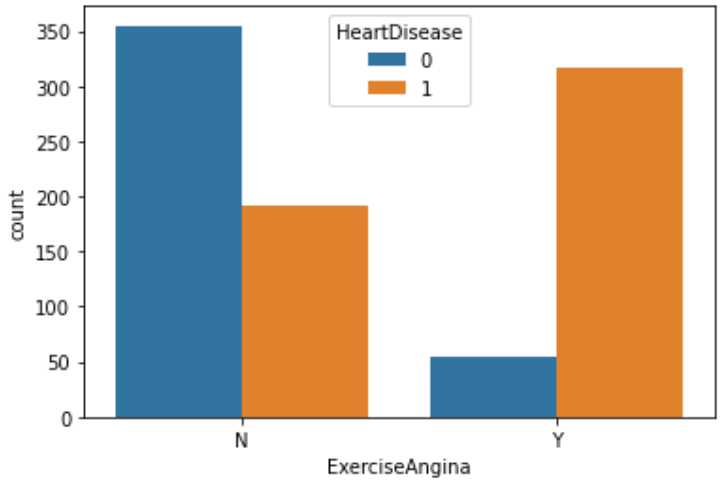
Observation:

Resting ECG are three types. For who have normal Resting ECG those people has affected more in Heart Disease compared with other two ST & LVH.

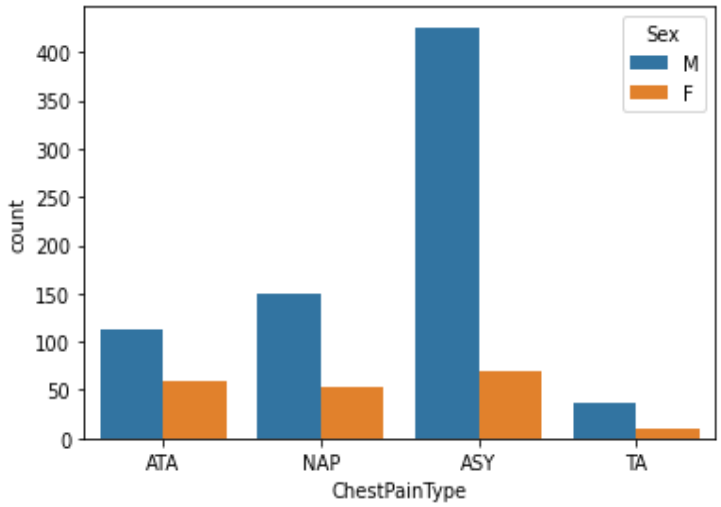
Bivariate Analysis of Exercise Angina

Observation:

- Who has Exercise Angina or who do not exercise, they are affected highly in Heart Disease
- Who doing exercise daily, they are affected in lesser amount



Bivariate Analysis between Chest Pain Type & Sex



Observation:

- There have four types of Chest pain, according to our dataset males are hugely affected by the ASY type & lesser affected by the TA type
- For the case of female, this affected rate is very minimum compared with the male

3. Data Cleaning & Pre-processing

Data Cleaning is an important phase in any data science project, if our data is clean then only we can provide it to our machine learning model. Un-cleaned Data can further lead our model with low accuracy. And, if data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset.

The approach used for identifying and treating missing values and outliertreatment:-

```
df.isnull().sum()
```

```
Age              0
Sex              0
ChestPainType    0
RestingBP        0
Cholesterol      0
FastingBS        0
RestingECG       0
MaxHR            0
ExerciseAngina   0
Oldpeak          0
ST_Slope         0
HeartDisease     0
dtype: int64
```

To identify any missing values in our data set we have used **Pandas** pre built function **isnull()** to detect any missing values in our datasets. In my data set there have no any missing values in any columns, but there have some non-numeric columns which are required for modeling purpose. So convert all those non-numeric columns into numeric columns by using dummies variable.

Variables removed or added and why?

Variables are removed in such a scenario where we have some categorical columns in our dataset, to make our data clean and ready for modeling. Whereas, variables are added in such a scenario where dummies variable columns are added in the data set.

Feature Scaling -

Feature scaling is important for every algorithm. Two famous techniques for Feature Scaling are:

1. Normalization

2. Standardization

Standardization is useful when the feature distribution is Normal or Gaussian, otherwise we do Normalization.

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
new = df2.select_dtypes(include="float64")
```

```
new_scaler = scaler.fit_transform(df2)
```

```
new_df = pd.DataFrame(new_scaler,columns=df2.columns)
```

```
new_df
```

	Sex_M	Age	MaxHR	RestingBP	Cholesterol	ChestPainType_ATA	ChestPainType_NAP	ChestPainType_TA	ExerciseAngina_Y	RestingECG_Normal	F
0	1.0	0.244898	0.788732	0.70	0.479270	1.0	0.0	0.0	0.0	1.0	
1	0.0	0.428571	0.676056	0.80	0.298507	0.0	1.0	0.0	0.0	1.0	
2	1.0	0.183673	0.267606	0.65	0.469320	1.0	0.0	0.0	0.0	0.0	
3	0.0	0.408163	0.338028	0.69	0.354892	0.0	0.0	0.0	1.0	1.0	
4	1.0	0.530612	0.436620	0.75	0.323383	0.0	1.0	0.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	
913	1.0	0.346939	0.507042	0.55	0.437811	0.0	0.0	1.0	0.0	1.0	
914	1.0	0.816327	0.570423	0.72	0.320066	0.0	0.0	0.0	0.0	1.0	
915	1.0	0.591837	0.387324	0.65	0.217247	0.0	0.0	0.0	1.0	1.0	
916	0.0	0.591837	0.802817	0.65	0.391376	1.0	0.0	0.0	0.0	0.0	
917	1.0	0.204082	0.795775	0.69	0.290216	0.0	1.0	0.0	0.0	1.0	

918 rows × 15 columns

4.Model Building

Model Selection and Why?
After cleaning and processing the data then comes the modeling part which includes building Machine Learning models, let's first understand in brief what Machine Learning is? Machine Learning is a technique that analyzes past data and tries to extract meaningful insights and patterns from them which can be further used to perform predictions in future. For example, classifying whether a tumor is benign or malignant, predicting stock prices,etc. One such application which we're using right here is predicting heart disease. Beforemaking predictions first we need to build a model and train it using past data.
First, we need to separate the dataset into two parts: features (attributes) and labels (target variable) which is the required format for any model to be trained on.
Then the data needs to be split into 3 sets

1. **Training set** - This will be the part of the dataset which the model will be using to train itself, the size should be at least 60-80% of the total data we've.
2. **Validation set** - This set is used for validating our model's performance for a different set of hyperparameters. After taking out the train set, the remaining set can be split into validation and test set.
3. **Testing set** - To evaluate how the model is performing on the unseen data on whichthe model will be doing future predictions on, test set is used. It helps to understandhow much error is there between actual and predicted values.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)
```

```
X_train.shape
```

```
(734, 14)
```

```
X_test.shape
```

```
(184, 14)
```

We need to build different classification algorithms and using the validation set we can determine which model to keep for making final predictions. Here is the list of all the algorithms we've to build and evaluated:

- K Nearest Neighbor classifier (KNN)
- Logistic Regression classifier
- Decision Tree classifier
- Random Forest classifier
- Support Vector Machine (SVM)classifier
- Gradient Boosting classifier
- XG Boosting classifier

Initially, we've tried KNN, SVM and Decision Tree but they are performing quite similarly on the 2 sets (train & test).

They are giving quite good result on train, but not good in test set, with KNN giving 64% accuracy for test set which is very poor.
Let's look at some other algorithms, and how they are performing as compared to KNN, SVM & Decision Tree. So, Logistic Regression classifier is giving quite good score of 0.871 on the train set, but not that good on the valid and test set, looks like itis over fitting on the training data

```
compare_df
```

	Classifier	Train Accuracy	Test Accuracy	Precision	Recall	F1 score	Accuracy
0	K Nearest Neighbor	0.820	0.652	0.695	0.695	0.695	0.645
1	Logistic Regression	0.871	0.826	0.848	0.848	0.848	0.823
2	Decision Tree	1.000	0.783	0.790	0.822	0.806	0.778
3	Random Forest	0.999	0.848	0.857	0.874	0.865	0.844
4	SVM	0.737	0.658	0.676	0.710	0.693	0.653
5	Gradient Boosting	0.935	0.870	0.895	0.879	0.887	0.868
6	XG Boosting	1.000	0.837	0.838	0.871	0.854	0.833

Finally, now we can try those boosting algorithms and see where they are getting us? Generally boosting algorithms give a very good performance, according to our data set Gradient Boosting classifier & XG Boosting classifier gave a good accuracy in both train & test set. Also Random Forest classifier works very well.

```
compare_df.sort_values(by=['Test Accuracy'], ascending=False)
```

	Classifier	Train Accuracy	Test Accuracy	Precision	Recall	F1 score	Accuracy
5	Gradient Boosting	0.935	0.870	0.895	0.879	0.887	0.868
3	Random Forest	0.999	0.848	0.857	0.874	0.865	0.844
6	XG Boosting	1.000	0.837	0.838	0.871	0.854	0.833
1	Logistic Regression	0.871	0.826	0.848	0.848	0.848	0.823
2	Decision Tree	1.000	0.783	0.790	0.822	0.806	0.778
4	SVM	0.737	0.658	0.676	0.710	0.693	0.653
0	K Nearest Neighbor	0.820	0.652	0.695	0.695	0.695	0.645

Efforts to improve model performance

Hyperparameter Tuning

Hyperparameter tuning is the process of trying out a different set of model parameters, Tuning hyperparameters helps us to find out the optimal parameter values for which the model is giving less error and a better overall accuracy score. We've performed tuning for Gradient Boosting Classifier with both the method **RandomizedSearchCV** as well as **GridSearchCV.** What random search does is from the given set of parameter values, it tries out **n** number of combinations whereas grid search builds a model with all the possible combinations from the given set of parameters and gives us the optimal values.

Then we need to provide it with the estimator and specify other things such as how many cross-validations sets to evaluate upon.

```
# to shrinks the contribution of each tree by learning_rate
learning_rates = [1, 0.5, 0.25, 0.1, 0.05, 0.01]

# Number of trees
n_estimators = [100,150,200]

# Maximum depth of trees
max_depth = [10,20,30]

# Minimum number of samples required to split a node
min_samples_split = [100,150,200]

# Minimum number of samples required at each leaf node
min_samples_leaf = [40,50]

# Hyperparameter Grid
param_dict = {'learning_rate': learning_rates,
              'n_estimators' : n_estimators,
              'max_depth' : max_depth,
              'min_samples_split' : min_samples_split,
              'min_samples_leaf' : min_samples_leaf}
```

```
# Create an instance of the Gradient BoostingClassifier
gbc = GradientBoostingClassifier(random_state=42)

# Grid search
gbc_grid = RandomizedSearchCV(estimator=gbc,
                      param_distributions = param_dict,
                      cv = 2, verbose=2, scoring='roc_auc')
# fitting model
gbc_grid.fit(X_train,y_train)
```

After fitting GridSearchCV it returns those params with which it gotthe best score.

```
gbc_grid.best_params_
```

```
{'n_estimators': 100,
 'min_samples_split': 150,
 'min_samples_leaf': 40,
 'max_depth': 10,
 'learning_rate': 0.1}
```

Getting all scores for Gradient Boosting after applying CV and Hyperparameter Tuning

```
#getting all scores for Gradient Boosting after CV and Hyperparameter Tunning
train_accuracy_gbc_grid = round(accuracy_score(y_train_pred_gbc_grid,y_train), 3)
accuracy_gbc_grid = round(accuracy_score(y_pred_gbc_grid,y_test), 3)
precision_score_gbc_grid = round(precision_score(y_pred_gbc_grid,y_test), 3)
recall_score_gbc_grid = round(recall_score(y_pred_gbc_grid,y_test), 3)
f1_score_gbc_grid = round(f1_score(y_pred_gbc_grid,y_test), 3)
auc_gbc_grid = round(roc_auc_score(y_pred_gbc_grid,y_test), 3)

print("The accuracy on train data is ", train_accuracy_gbc_grid)
print("The accuracy on test data is ", accuracy_gbc_grid)
print("The precision on test data is ", precision_score_gbc_grid)
print("The recall on test data is ", recall_score_gbc_grid)
print("The f1 on test data is ", f1_score_gbc_grid)
print("The auc on test data is ", auc_gbc_grid)
```

```
The accuracy on train data is  0.944
The accuracy on test data is  0.864
The precision on test data is  0.876
The recall on test data is  0.885
The f1 on test data is  0.88
The auc on test data is  0.861
```

5. Model Validation

Performance Metrics

Just building is not enough, as we need to evaluate it using differentmetrics based on the problem we're solving. Model Validation helps us to understand how well the model is generalizing on the real- world data, the data which it has not seen during the training phase.

For classification problems the evaluation metrics we've used are:

- Confusion matrix
- Classification report

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**.

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report.

6. Final Conclusions

I have applied seven different types of classification algorithm on my given dataset to know which algorithm good fit for our dataset & gives us the best accuracy. Before applying Cross validation and hyperparameter tuning Gradient Boosting shows highest test accuracy score of 0.870, F1 score is 0.887 and Accuracy is 0.868, but after applying Cross validation and hyperparameter tuning on the Gradient Boosting algorithm it gives test accuracy score of 0.864, F1 score is 0.88 and Accuracy is 0.861 which is almost same than before. As we know that Cross validation and hyperparameter tuning certainly reduces chances of overfitting and also increases performance of model. So we can conclude that before tuning our model worked well so here no needed to apply tuning as such.