MACHINE LEARNING

CSE4036

PROJECT REPORT

TOPIC: HEART DISEASE PREDICTION

TEAM MEMEBERS

HARINI GOKULRAM NAIDU 19MIA1004

P SUBHASHRI 19MIA1008

ANNUGRAHA S 19MIA1059

FACULTY

Dr. BHARGAVI R

ACKNOWLEDGEMENT

We express our sincere gratitude to all those who have been helpful by being a part in the completion of this project. We are also extremely thankful to Dr R Bhargavi, Associate Senior Professor, Vellore Institute of Technology, Chennai., for the guidance and support without which the completion of the project would not have been possible.

ABSTRACT

The major challenge in heart disease is its detection. There are instruments available which can predict heart disease but either it is expensive or are not efficient to calculate chance of heart disease in human. Early detection of cardiac diseases can decrease the mortality rate and overall complications. However, it is not possible to monitor patients every day in all cases accurately and consultation of a patient for 24 hours by a doctor is not available since it requires more sapience, time and expertise. Since we have a good amount of data in today’s world, we can use various machine learning algorithms to analyze the data for hidden patterns. The hidden patterns can be used for health diagnosis in medicinal data.

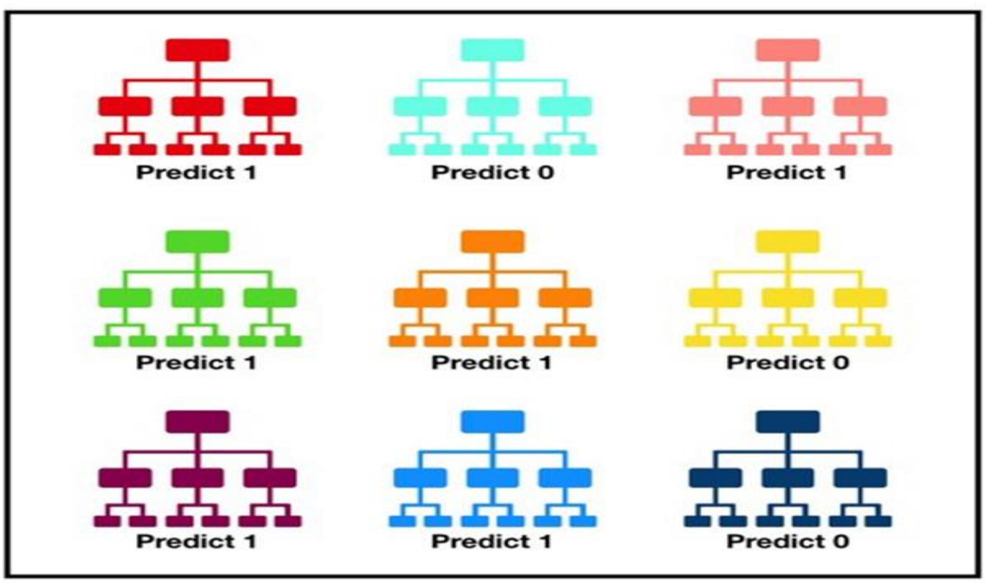
INTRODUCTION

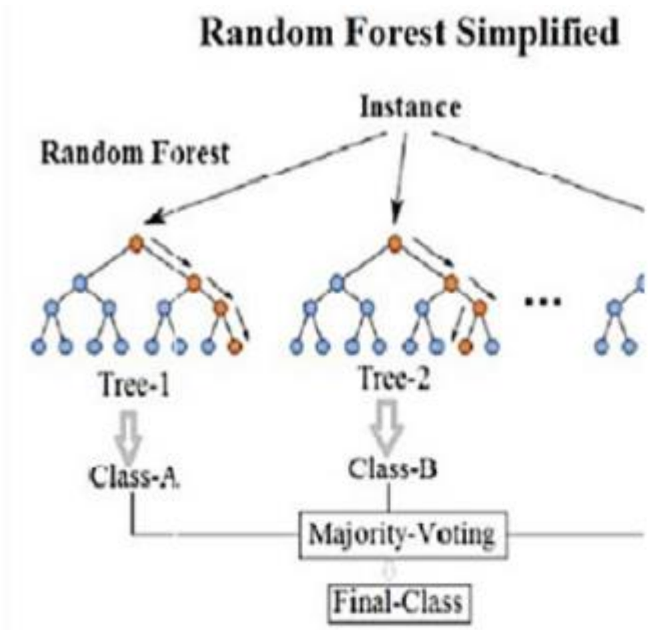
Heart-related diseases or cardiovascular diseases (CVDs) are the main reason for a huge number of deaths in the world over the last few decades and has emerged as the most life-threatening disease, not only in India but in the whole world. So, there is a need for a reliable, accurate, and feasible system to diagnose such diseases in time for proper treatment. Machine Learning algorithms and techniques have been applied to various medical datasets to automate the analysis of large and complex data. Many researchers, in recent times, have been using several machine learning techniques to help the health care industry and the professionals in the diagnosis of heart-related diseases. Heart is the next major organ comparing to the brain which has more priority in the Human body. It pumps the blood and supplies it to all organs of the whole body. Prediction of occurrences of heart diseases in the medical field is significant work. Data analytics is useful for prediction from more information and it helps the medical center to predict various diseases. A huge amount of patient-related data is maintained on monthly basis. The stored data can be useful for the source of predicting the occurrence of future diseases. Some of the data mining and machine learning techniques are used to predict heart diseases, such as Random Forest and Support Vector Machine (SVM). Prediction and diagnosing of heart disease become a challenging factor faced by doctors and hospitals both in India and abroad. To reduce the large scale of deaths from heart diseases, a quick and efficient detection technique is to be discovered. Machine learning algorithms play a very important role in this area. The researchers accelerating their research work to develop software with the help of machine learning algorithms which can help doctors to decide both prediction and diagnosing of heart disease. The main objective of this project is to check which algorithm has the highest accuracy.

DESCRIPTION OF MODULES USED

The Random Forest Classifier

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.





Visualization of a Random Forest Model Making a Prediction

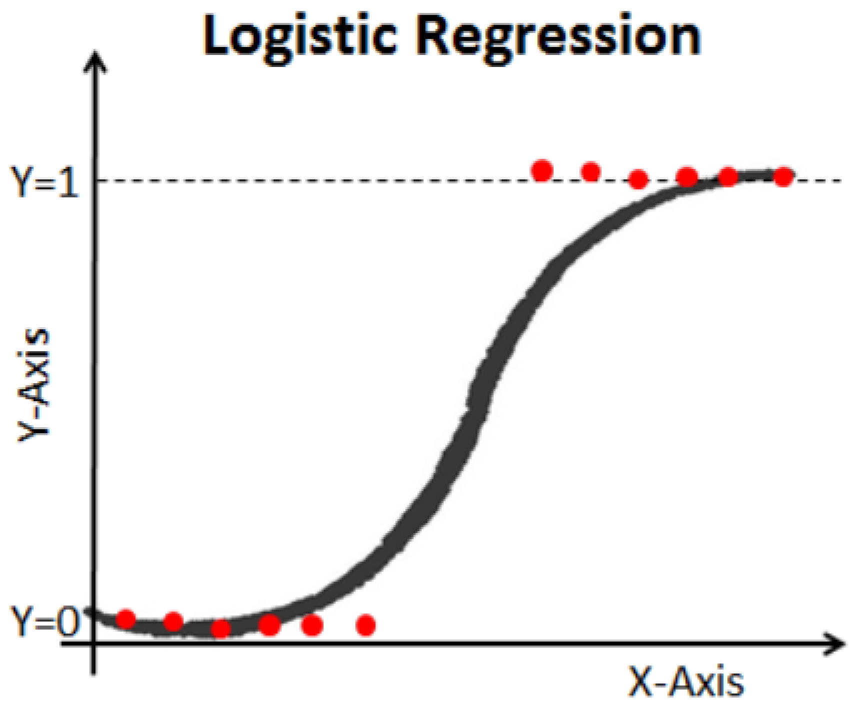
The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. The reason that the random forest model works so well is: A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.

2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

LOGISTIC REGRESSION:

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. Logistic Regression is another statistical analysis method borrowed by Machine Learning. It is used when our dependent variable is dichotomous or binary. It just means a variable that has only 2 outputs, for example, A person will survive this accident or not, The student will pass this exam or not. The outcome can either be yes or no (2 outputs). This regression technique is similar to linear regression and can be used to predict the Probabilities for classification problems.

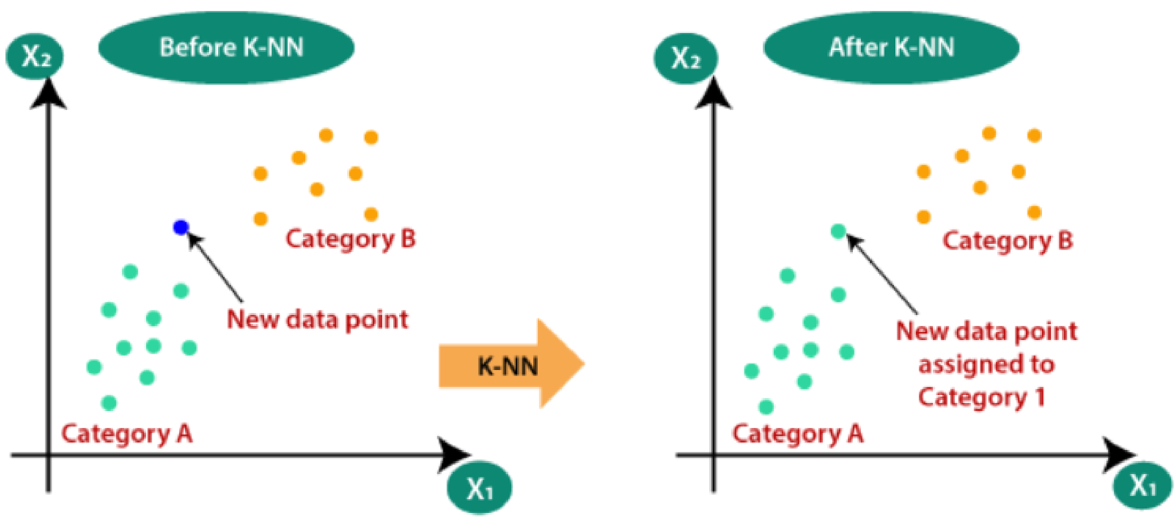


K-Nearest Neighbors

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. 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. KNN algorithm at the training phase just stores the dataset and when it gets new data, and then it classifies that data into a category that is much similar to the new data.

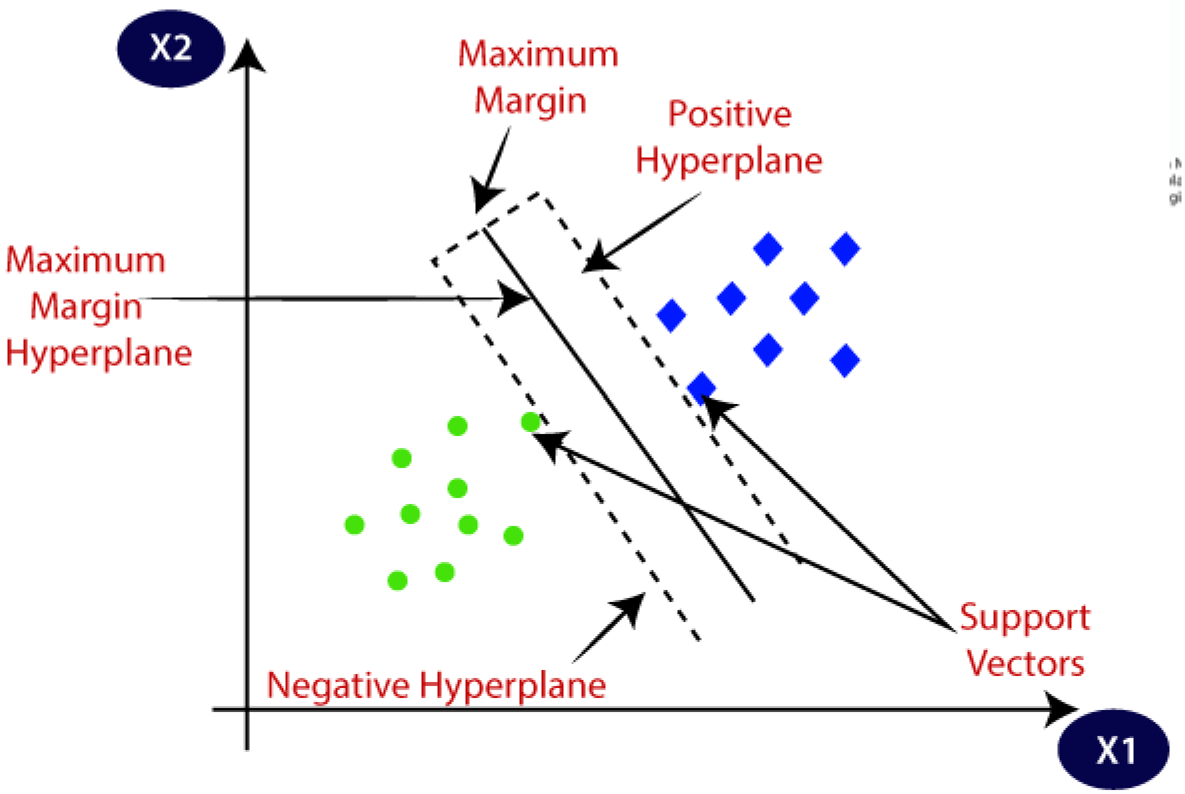
Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.



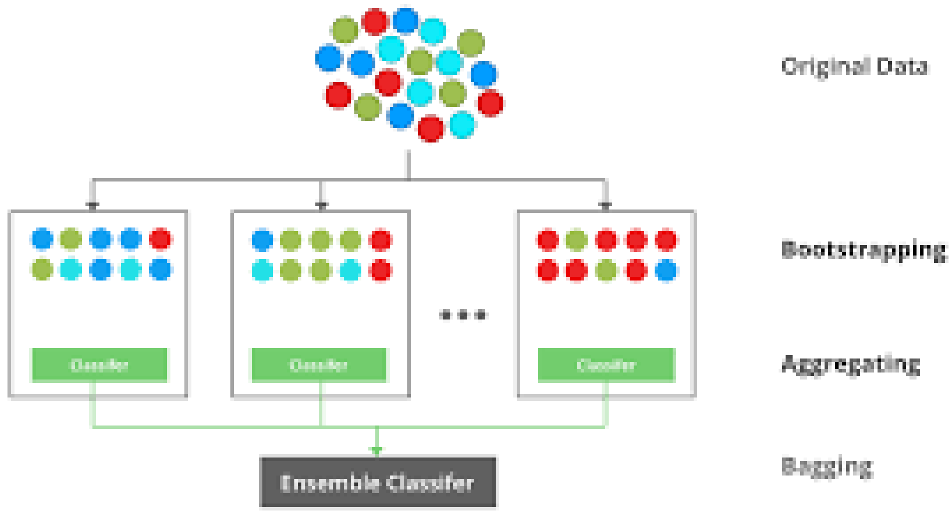
Support vector Machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.



XG Boost Classifier

XGBoost provides a wrapper class to allow models to be treated like classifiers or regressors in the scikit-learn framework. This means we can use the full scikit-learn library with XGBoost models. The XGBoost model for classification is called XGBClassifier. We can create and fit it to our training dataset. Models are fit using the scikit-learn API and the model.fit() function. Parameters for training the model can be passed to the model in the constructor. Here, we use the sensible defaults.



Decision Tree

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, like a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data

Model: Decision Tree

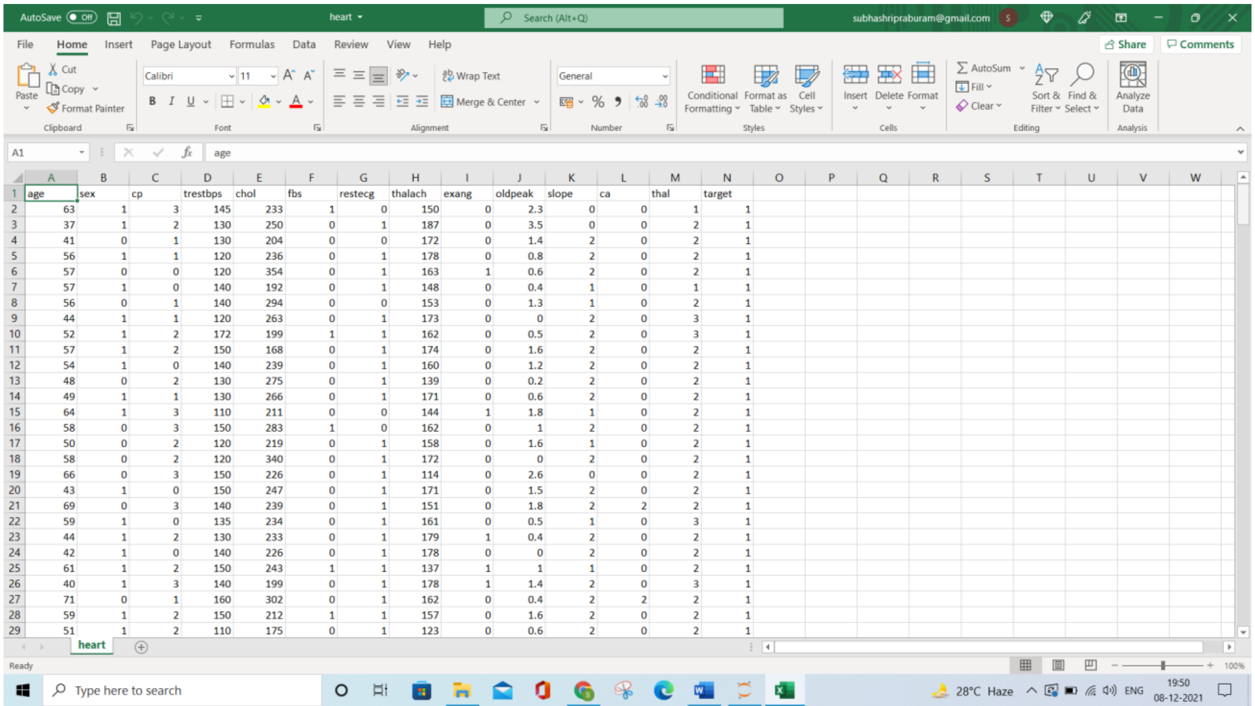
DATASET DESCRIPTION

The dataset used here is taken from a UCI repository which contains various instances with different attributes of some of the patients.

This dataset has 14 attributes in it. Let’s see about those in detail.

- **age:** age in years.
- **sex:** sex (1 = male; 0 = female).
- **cp:** chest pain type (Value 0: typical angina; Value 1: atypical angina; Value 2: non-anginal pain; Value 3: asymptomatic).
- **trestbps:** resting blood pressure in mm Hg on admission to the hospital.
- **chol:** serum cholesterol in mg/dl.
- **fbs:** fasting blood sugar > 120 mg/dl (1 = true; 0 = false).
- **restecg:** resting electrocardiographic results (Value 0: normal; Value 1: having ST-T wave abnormality; Value 2: probable or definite left ventricular hypertrophy).
- **thalach:** maximum heart rate achieved.
- **exang:** exercise induced angina (1 = yes; 0 = no)
- **old peak:** ST depression induced by exercise relative to rest.
- **slope:** the slope of the peak exercise ST segment (Value 0: upsloping; Value 1: flat; Value 2: down sloping).
- **ca:** number of major vessels (0-3) coloured by fluoroscopy.
- **thal:** Thalassemia (3 = normal; 6 = fixed defect; 7 = reversable defect).

- **target:** Has a Heart disease(1 = no, 2 = yes)



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
2	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
3	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
4	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
5	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
6	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
7	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
8	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
9	44	1	1	120	263	0	1	173	0	0	2	0	3	1
10	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
11	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
12	54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
13	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
14	49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
15	64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
16	58	0	3	150	283	1	0	162	0	1	2	0	2	1
17	50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
18	58	0	2	120	340	0	1	172	0	0	2	0	2	1
19	66	0	3	150	226	0	1	114	0	2.6	0	0	2	1
20	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1
21	69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
22	59	1	0	135	234	0	1	161	0	0.5	1	0	3	1
23	44	1	2	130	233	0	1	179	1	0.4	2	0	2	1
24	42	1	0	140	226	0	1	178	0	0	2	0	2	1
25	61	1	2	150	243	1	1	137	1	1	1	0	2	1
26	40	1	3	140	199	0	1	178	1	1.4	2	0	3	1
27	71	0	1	160	302	0	1	162	0	0.4	2	2	2	1
28	59	1	2	150	212	1	1	157	0	1.6	2	0	2	1
29	51	1	2	110	175	0	1	123	0	0.6	2	0	2	1

CODE AND SCREENSHOTS:

IMPORTING THE DEPENDENCIES

```
[ ]
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

DATA COLLECTION AND PROCESSING

```
[ ]
# loading the csv data to a Pandas DataFrame
heart_data = pd.read_csv('/content/data.csv')
```

```
[ ]
# print first 5 rows of the dataset
heart_data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1

+ Code + Text Connect Editing

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
# print last 5 rows of the dataset
heart_data.tail()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

```
# number of rows and columns in the dataset
heart_data.shape
```

+ Code + Text Connect Editing

```
# number of rows and columns in the dataset
heart_data.shape
```

```
(303, 14)
```

```
# getting some info about the data
heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 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
```

+ Code + Text Connect Editing

```
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
# checking for missing values
heart_data.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

```
# statistical measures about the data
heart_data.describe()
```

+ Code + Text Connect Editing

[ ]

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

```
# checking the distribution of Target Variable
heart_data['target'].value_counts()
```

```
1    165
0    138
Name: target, dtype: int64
```

1 --> Defective Heart

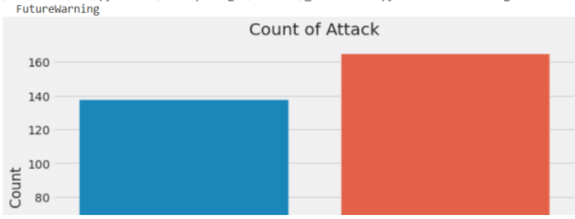
0 --> Healthy Heart

+ Code + Text Connect Editing

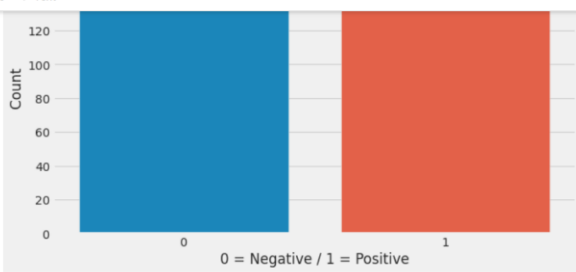
```
heart_data_0 = heart_data[(heart_data['target'] == 0)]
heart_data_1 = heart_data[(heart_data['target'] == 1)]
```

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(rc={'figure.figsize':(10,6)})
sns.set(font_scale=1.3)
plt.style.use('fivethirtyeight')
ax = sns.countplot(heart_data['target'])
ax.set_title('Count of Attack');
ax.set(xlabel='0 = Negative / 1 = Positive', ylabel='Count');
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument
  FutureWarning
```

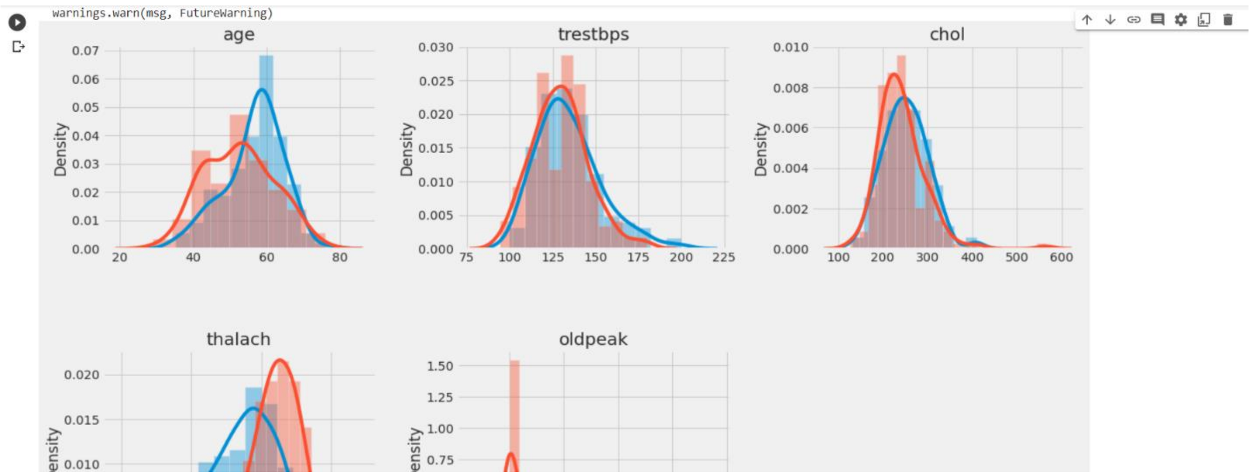


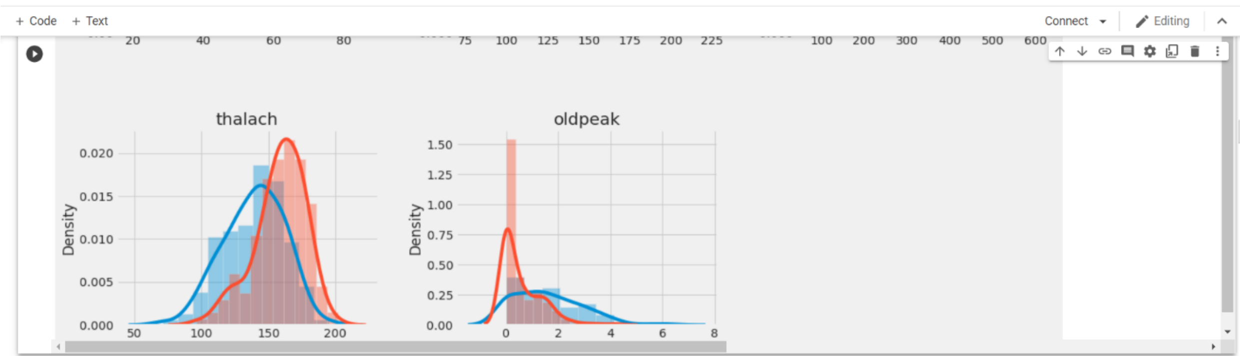
+ Code + Text Connect Editing




```
columns = ['age','trestbps','chol','thalach','oldpeak']

length = len(columns)
plt.subplots(figsize=(18,10))

for i, j in zip(columns, range(length)):
    plt.subplot((length/2), 3, j+1)
    plt.subplots_adjust(wspace=.3, hspace=.5)
    sns.distplot(x=heart_data_0[i])
```

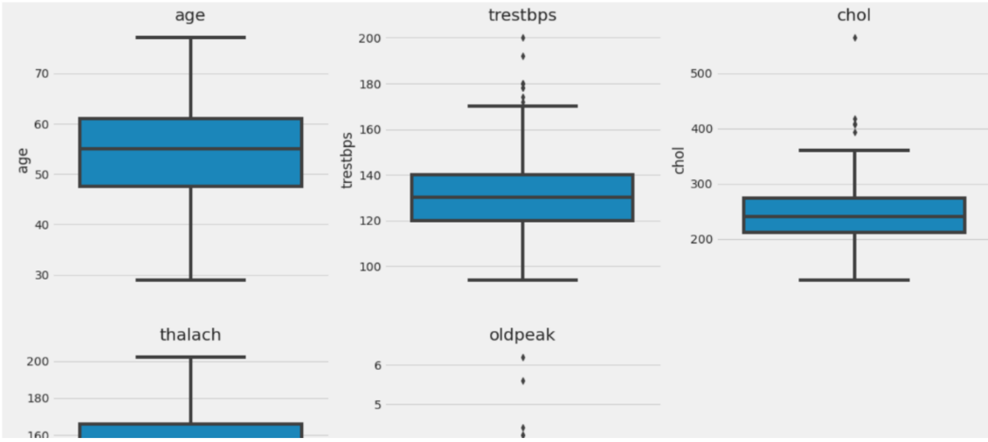
```python
columns = ['age','trestbps','chol','thalach','oldpeak']

length = len(columns)
plt.subplots(figsize=(18,10))

for i, j in zip(columns, range(length)):
    plt.subplot((length/2), 3, j+1)
    plt.subplots_adjust(wspace=.3, hspace=.5)
    sns.distplot(x=heart_data_0[i])
    sns.distplot(x=heart_data_1[i])
    plt.title(i);
```



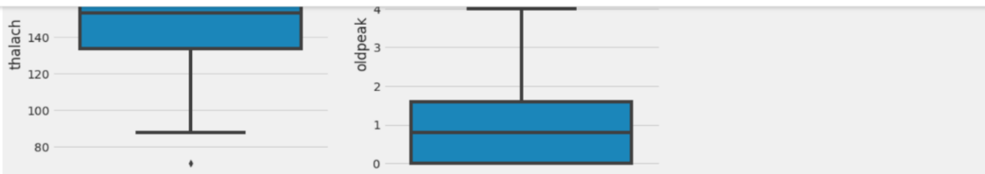



```python
plt.subplots(figsize=(18,12))
length = len(columns)

for i, j in zip(columns, range(length)):
    plt.subplot((length/2), 3, j+1)
    sns.boxplot(y=heart_data[i])
    plt.title(i)
```

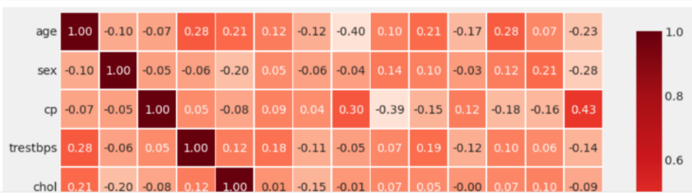
+ Code + Text Connect Editing



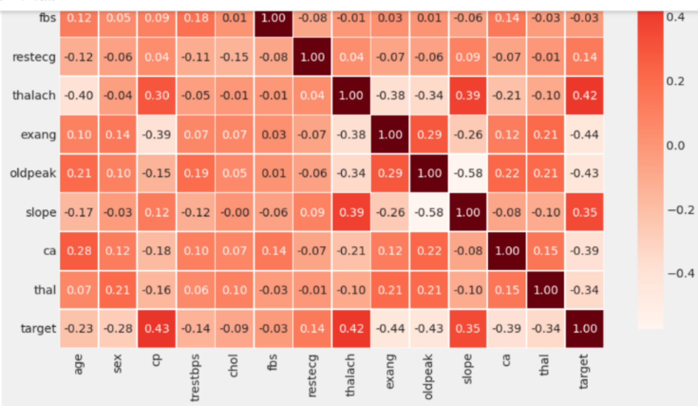
+ Code + Text Connect Editing




```
plt.figure(figsize=(13,13))
sns.heatmap(heart_data.corr(), annot=True, cmap=plt.cm.Reds, cbar_kws={'shrink': .8}, square=True, fmt='.2f',
            linewidths=.8);
```



+ Code + Text Connect Editing



SPLITTING THE FEATURES AND TARGET

+ Code + Text Connect Editing

```
[ ]  X = heart_data.drop(columns='target', axis=1)
     Y = heart_data['target']
```

```
[ ]  print(X)
```

```
     age  sex  cp  trestbps  chol  ...  exang  oldpeak  slope  ca  thal
0     63    1   3       145   233  ...      0      2.3      0   0     1
1     37    1   2       130   250  ...      0      3.5      0   0     2
2     41    0   1       130   204  ...      0      1.4      2   0     2
3     56    1   1       120   236  ...      0      0.8      2   0     2
4     57    0   0       120   354  ...      1      0.6      2   0     2
..   ...  ...  ..       ...   ...  ...    ...      ...    ...  ..   ...
298   57    0   0       140   241  ...      1      0.2      1   0     3
299   45    1   3       110   264  ...      0      1.2      1   0     3
300   68    1   0       144   193  ...      0      3.4      1   2     3
301   57    1   0       130   131  ...      1      1.2      1   1     3
302   57    0   1       130   236  ...      0      0.0      1   1     2

[303 rows x 13 columns]
```

```
[ ]  print(Y)
```

```
0      1
1      1
2      1
3      1
4      1
```

+ Code + Text Connect Editing

```
[ ]  298    0
     299    0
     300    0
     301    0
     302    0
     Name: target, Length: 303, dtype: int64
```

SPLITTING THE DATA INTO TRAINING AND TESTING DATA

```
[ ]  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

```
[ ]  print(X.shape, X_train.shape, X_test.shape)
```

```
(303, 13) (242, 13) (61, 13)
```

MODEL TRAINING

```
[ ]  from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

     def print_score(clf, X_train, y_train, X_test, y_test, train=True):
         if train:
             pred = clf.predict(X_train)
             clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
             print("Train Result:\n================================================")
             print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
```

+ Code + Text Connect Editing

```
[ ]          print("_______________________________________________")
             print(f"CLASSIFICATION REPORT:\n{clf_report}")
             print("_______________________________________________")
             print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

         elif train==False:
             pred = clf.predict(X_test)
             clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
             print("Test Result:\n================================================")
             print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
             print("_______________________________________________")
             print(f"CLASSIFICATION REPORT:\n{clf_report}")
             print("_______________________________________________")
             print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

RANDOM FOREST CLASSIFIER

```
[ ]  from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
     randomforest_classifier= RandomForestClassifier(n_estimators=10)
     randomforest_classifier.fit(X_train, Y_train)

     print_score(randomforest_classifier, X_train, Y_train, X_test, Y_test, train=True)
     print_score(randomforest_classifier, X_train, Y_train, X_test, Y_test, train=False)

```

+ Code + Text Connect Editing

```
Train Result:
================================================
Accuracy Score: 99.59%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    1.000000    0.992481  0.995868    0.996241      0.995899
recall       0.990909    1.000000  0.995868    0.995455      0.995868
f1-score     0.995434    0.996226  0.995868    0.995830      0.995866
support    110.000000  132.000000  0.995868  242.000000    242.000000
_______________________________________________
Confusion Matrix:
 [[109   1]
 [  0 132]]

Test Result:
================================================
Accuracy Score: 70.49%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.678571   0.727273  0.704918   0.702922      0.704918
recall      0.678571   0.727273  0.704918   0.702922      0.704918
f1-score    0.678571   0.727273  0.704918   0.702922      0.704918
support    28.000000  33.000000  0.704918  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[19  9]
 [ 9 24]]
```

+ Code + Text Connect Editing

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
randomforest_classifier= RandomForestClassifier(n_estimators=400,max_depth=30, random_state=92)
randomforest_classifier.fit(X_train, Y_train)

print_score(randomforest_classifier, X_train, Y_train, X_test, Y_test, train=True)
print_score(randomforest_classifier, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    110.0  132.0       1.0      242.0         242.0
_______________________________________________
Confusion Matrix:
 [[110   0]
 [  0 132]]

Test Result:
================================================
Accuracy Score: 77.05%
_______________________________________________
CLASSIFICATION REPORT:
```

+ Code + Text Connect Editing

```
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.733333   0.806452  0.770492   0.769892      0.772889
recall      0.785714   0.757576  0.770492   0.771645      0.770492
f1-score    0.758621   0.781250  0.770492   0.769935      0.770863
support    28.000000  33.000000  0.770492  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[22  6]
 [ 8 25]]
```

```python
model_evaluation = cross_val_score(estimator = randomforest_classifier , X = X, y = Y, cv = 15)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

```
Model evaluation scores: [0.9047619  0.85714286 0.80952381 0.85       0.9        0.95
 0.85       0.85       0.75       0.9        0.7        0.85
 0.7        0.8        0.8       ]
Model Evaluation mean score: 0.8314285714285714
```

```python
model_evaluation = cross_val_score(estimator = randomforest_classifier , X = X, y = Y, cv = 10)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

```
Model evaluation scores: [0.90322581 0.80645161 0.87096774 0.9        0.9        0.83333333
 0.7        0.83333333 0.73333333 0.76666667]
Model Evaluation mean score: 0.824731182795699
```

+ Code + Text Connect Editing

LOGISTIC REGRESSION

```
[ ]
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression(C=100, random_state=20, solver='lbfgs')
lr_clf.fit(X_train, Y_train)

print_score(lr_clf, X_train, Y_train, X_test, Y_test, train=True)
print_score(lr_clf, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 85.54%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.894737    0.829932  0.855372    0.862334      0.859389
recall       0.772727    0.924242  0.855372    0.848485      0.855372
f1-score     0.829268    0.874552  0.855372    0.851910      0.853968
support    110.000000  132.000000  0.855372  242.000000    242.000000
_______________________________________________
Confusion Matrix:
 [[ 85  25]
 [ 10 122]]
```

+ Code + Text Connect Editing

```
[ ]
Test Result:
================================================
Accuracy Score: 81.97%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.793103   0.843750  0.819672   0.818427      0.820502
recall      0.821429   0.818182  0.819672   0.819805      0.819672
f1-score    0.807018   0.830769  0.819672   0.818893      0.819867
support    28.000000  33.000000  0.819672  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[23  5]
 [ 6 27]]
```

+ Code + Text Connect Editing

```
[ ]
lr_clf1 = LogisticRegression(C=230, random_state=40)
lr_clf1.fit(X_train, Y_train)

print_score(lr_clf1, X_train, Y_train, X_test, Y_test, train=True)
print_score(lr_clf1, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 85.54%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.894737    0.829932  0.855372    0.862334      0.859389
recall       0.772727    0.924242  0.855372    0.848485      0.855372
f1-score     0.829268    0.874552  0.855372    0.851910      0.853968
support    110.000000  132.000000  0.855372  242.000000    242.000000
_______________________________________________
Confusion Matrix:
 [[ 85  25]
 [ 10 122]]

Test Result:
================================================
Accuracy Score: 83.61%
_______________________________________________
```

+ Code + Text Connect Editing

```
[ ]
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.821429   0.848485  0.836066   0.834957      0.836066
recall      0.821429   0.848485  0.836066   0.834957      0.836066
f1-score    0.821429   0.848485  0.836066   0.834957      0.836066
support    28.000000  33.000000  0.836066  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[23  5]
 [ 5 28]]
```

```
[ ]
model_evaluation = cross_val_score(estimator = lr_clf , X = X, y = Y, cv = 10)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

+ Code + Text Connect Editing

K-NEAREST NEIGHBOURS

```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')
knn_clf.fit(X_train, Y_train)

print_score(knn_clf, X_train, Y_train, X_test, Y_test, train=True)
print_score(knn_clf, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 78.10%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.771429    0.788321  0.780992    0.779875      0.780643
recall       0.736364    0.818182  0.780992    0.777273      0.780992
f1-score     0.753488    0.802974  0.780992    0.778231      0.780481
support    110.000000  132.000000  0.780992  242.000000    242.000000
_______________________________________________
Confusion Matrix:
 [[ 81  29]
 [ 24 108]]
```

+ Code + Text Connect Editing

```
Test Result:
================================================
Accuracy Score: 62.30%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.592593   0.647059  0.622951   0.619826      0.622058
recall      0.571429   0.666667  0.622951   0.619048      0.622951
f1-score    0.581818   0.656716  0.622951   0.619267      0.622337
support    28.000000  33.000000  0.622951  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[16 12]
 [11 22]]
```

```
knn_clf1 = KNeighborsClassifier(n_neighbors=4, p=1, metric='minkowski')
knn_clf1.fit(X_train, Y_train)

print_score(knn_clf1, X_train, Y_train, X_test, Y_test, train=True)
print_score(knn_clf1, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 77.69%
_______________________________________________
```

+ Code + Text Connect Editing

```
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.705882    0.867925   0.77686    0.786903      0.794269
recall       0.872727    0.696970   0.77686    0.784848      0.776860
f1-score     0.780488    0.773109   0.77686    0.776799      0.776463
support    110.000000  132.000000   0.77686  242.000000    242.000000
_______________________________________________
Confusion Matrix:
 [[96 14]
 [40 92]]

Test Result:
================================================
Accuracy Score: 59.02%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.545455   0.642857  0.590164   0.594156      0.598148
recall      0.642857   0.545455  0.590164   0.594156      0.590164
f1-score    0.590164   0.590164  0.590164   0.590164      0.590164
support    28.000000  33.000000  0.590164  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[18 10]
 [15 18]]
```

+ Code + Text Connect Editing

```
model_evaluation = cross_val_score(estimator = knn_clf , X = X, y = Y, cv = 5)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

```
Model evaluation scores: [0.60655738 0.6557377  0.57377049 0.73333333 0.65      ]
Model Evaluation mean score: 0.643879781420765
```

```
model_evaluation = cross_val_score(estimator = knn_clf , X = X, y = Y, cv =12)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

```
Model evaluation scores: [0.73076923 0.65384615 0.46153846 0.72       0.64       0.72
 0.52       0.76       0.72       0.6        0.6        0.8       ]
Model Evaluation mean score: 0.6605128205128205
```

SVM

```
from sklearn.svm import SVC


svm_clf = SVC(kernel='rbf', C=0.783)
svm_clf.fit(X_train, Y_train)
```

+ Code + Text Connect Editing

```
svm_clf.fit(X_train, Y_train)

print_score(svm_clf, X_train, Y_train, X_test, Y_test, train=True)
print_score(svm_clf, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 67.77%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.728571    0.656977  0.677686    0.692774      0.689520
recall       0.463636    0.856061  0.677686    0.659848      0.677686
f1-score     0.566667    0.743421  0.677686    0.655044      0.663078
support    110.000000  132.000000  0.677686  242.000000    242.000000
_______________________________________________
Confusion Matrix:
 [[ 51  59]
 [ 19 113]]

Test Result:
================================================
Accuracy Score: 63.93%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.650000   0.634146  0.639344   0.642073      0.641423
recall      0.464286   0.787879  0.639344   0.626082      0.639344
f1-score    0.541667   0.702703  0.639344   0.622185      0.628785
support    28.000000  33.000000  0.639344  61.000000     61.000000
```

+ Code + Text Connect Editing

```
_______________________________________________
Confusion Matrix:
 [[13 15]
 [ 7 26]]
```

```
svm_clf1 = SVC(kernel='linear', C=0.942)
svm_clf1.fit(X_train, Y_train)

print_score(svm_clf1, X_train, Y_train, X_test, Y_test, train=True)
print_score(svm_clf1, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 85.54%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.912088    0.821192  0.855372    0.866640      0.862508
recall       0.754545    0.939394  0.855372    0.846970      0.855372
f1-score     0.825871    0.876325  0.855372    0.851098      0.853391
support    110.000000  132.000000  0.855372  242.000000    242.000000
_______________________________________________
Confusion Matrix:
 [[ 83  27]
 [  8 124]]

Test Result:
================================================
Accuracy Score: 80.33%
```

+ Code + Text Connect Editing

```
Test Result:
================================================
Accuracy Score: 80.33%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.833333   0.783784  0.803279   0.808559      0.806528
recall      0.714286   0.878788  0.803279   0.796537      0.803279
f1-score    0.769231   0.828571  0.803279   0.798901      0.801333
support    28.000000  33.000000  0.803279  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[20  8]
 [ 4 29]]
```

```python
model_evaluation = cross_val_score(estimator = svm_clf , X = X, y = Y, cv =2)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

```
Model evaluation scores: [0.64473684 0.62913907]
Model Evaluation mean score: 0.6369379574764726
```

```python
model_evaluation = cross_val_score(estimator = svm_clf , X = X, y = Y, cv =6)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

+ Code + Text Connect Editing

```
Model evaluation scores: [0.62745098 0.56862745 0.74509804 0.64       0.66       0.6       ]
Model Evaluation mean score: 0.6401960784313726
```

XGBOOST CLASSIFIER

```python
from xgboost import XGBClassifier

xgb_clf = XGBClassifier(use_label_encoder=False)
xgb_clf.fit(X_train, Y_train)

print_score(xgb_clf, X_train, Y_train, X_test, Y_test, train=True)
print_score(xgb_clf, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 98.76%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    1.000000    0.977778  0.987603    0.988889      0.987879
recall       0.972727    1.000000  0.987603    0.986364      0.987603
f1-score     0.986175    0.988764  0.987603    0.987470      0.987587
support    110.000000  132.000000  0.987603  242.000000    242.000000
_______________________________________________
Confusion Matrix:
 [[107   3]
 [  0 132]]
```

+ Code + Text Connect Editing

```
Test Result:
================================================
Accuracy Score: 75.41%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.724138   0.781250  0.754098   0.752694      0.755035
recall      0.750000   0.757576  0.754098   0.753788      0.754098
f1-score    0.736842   0.769231  0.754098   0.753036      0.754364
support    28.000000  33.000000  0.754098  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[21  7]
 [ 8 25]]
```

```python
model_evaluation = cross_val_score(estimator = xgb_clf , X = X, y = Y, cv =6)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

```
Model evaluation scores: [0.88235294 0.80392157 0.92156863 0.72       0.78       0.74       ]
Model Evaluation mean score: 0.8079738562091504
```

```python
model_evaluation = cross_val_score(estimator = xgb_clf , X = X, y = Y, cv =14)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

+ Code + Text Connect Editing

```
Model evaluation scores: [0.86363636 0.81818182 0.77272727 0.90909091 0.81818182 0.86363636
 0.95454545 0.81818182 0.68181818 0.71428571 0.85714286 0.66666667
 0.76190476 0.76190476]
Model Evaluation mean score: 0.804421768707483
```

DECISION TREE

```python
from sklearn.tree import DecisionTreeClassifier


tree_clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=54)
tree_clf.fit(X_train, Y_train)

print_score(tree_clf, X_train, Y_train, X_test, Y_test, train=True)
print_score(tree_clf, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 85.95%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.839286    0.876923  0.859504    0.858104      0.859815
recall       0.854545    0.863636  0.859504    0.859091      0.859504
f1-score     0.846847    0.870229  0.859504    0.858538      0.859601
support    110.000000  132.000000  0.859504  242.000000    242.000000
```

+ Code + Text Connect Editing

```
_______________________________________________
Confusion Matrix:
 [[ 94  16]
 [ 18 114]]

Test Result:
================================================
Accuracy Score: 75.41%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.709677   0.800000  0.754098   0.754839      0.758540
recall      0.785714   0.727273  0.754098   0.756494      0.754098
f1-score    0.745763   0.761905  0.754098   0.753834      0.754495
support    28.000000  33.000000  0.754098  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[22  6]
 [ 9 24]]
```

```python
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=8, random_state=62)
tree_clf.fit(X_train, Y_train)

print_score(tree_clf, X_train, Y_train, X_test, Y_test, train=True)
print_score(tree_clf, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%


_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    110.0  132.0       1.0      242.0         242.0
_______________________________________________
Confusion Matrix:
 [[110   0]
 [  0 132]]

Test Result:
================================================
Accuracy Score: 75.41%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.740741   0.764706  0.754098   0.752723      0.753705
recall      0.714286   0.787879  0.754098   0.751082      0.754098
f1-score    0.727273   0.776119  0.754098   0.751696      0.753698
support    28.000000  33.000000  0.754098  61.000000     61.000000
_______________________________________________
Confusion Matrix:
 [[20  8]
 [ 7 26]]
```

+ Code + Text Connect Editing

```
model_evaluation = cross_val_score(estimator = tree_clf , X = X, y = Y, cv =11)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

```
Model evaluation scores: [0.85714286 0.78571429 0.78571429 0.82142857 0.82142857 0.67857143
 0.55555556 0.77777778 0.7037037  0.74074074 0.74074074]
Model Evaluation mean score: 0.7516835016835017
```

```
model_evaluation = cross_val_score(estimator = tree_clf , X = X, y = Y, cv =16)
print("Model evaluation scores:",model_evaluation)
print("Model Evaluation mean score:",model_evaluation.mean())
```

```
Model evaluation scores: [0.84210526 0.84210526 0.89473684 0.84210526 0.84210526 0.78947368
 0.84210526 0.78947368 0.78947368 0.57894737 0.73684211 0.63157895
 0.84210526 0.73684211 0.73684211 0.83333333]
Model Evaluation mean score: 0.7856359649122806
```

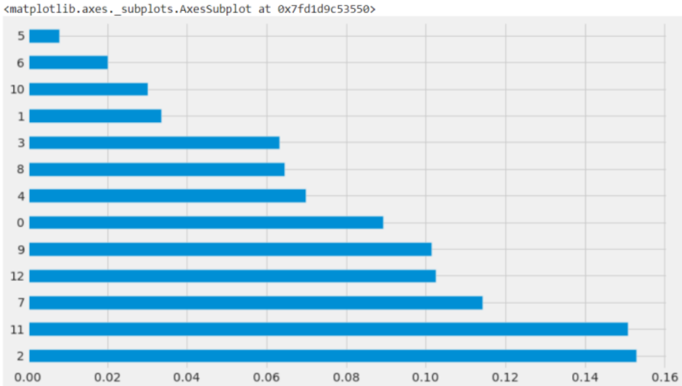
Features Importance According to Random Forest and XGBoost and Decision tree classifier

```
def feature_imp(df, model):
    fi = pd.DataFrame()
    fi["feature"] = df.columns
    fi["importance"] = model.feature_importances_
    return fi.sort_values(by="importance", ascending=False)
```

+ Code + Text Connect Editing

```
feature_imp(X, randomforest_classifier).plot(kind='barh', figsize=(12,7), legend=False)
```

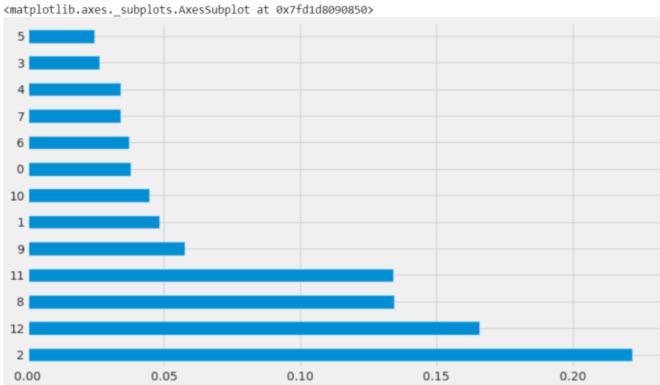
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1d9c53550>
```



+ Code + Text Connect Editing

```
feature_imp(X, xgb_clf).plot(kind='barh', figsize=(12,7), legend=False)
```

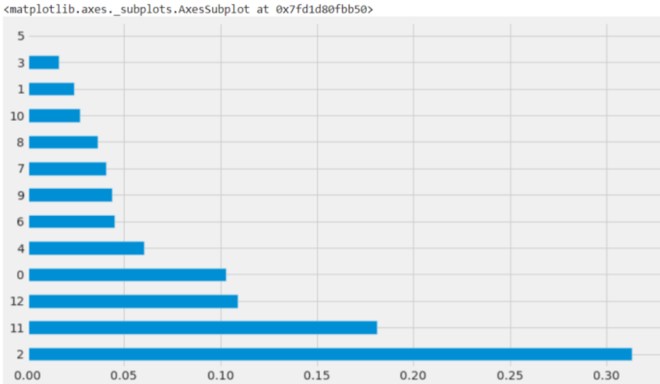
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1d8090850>
```



+ Code + Text Connect Editing

```
[ ] 1 feature_imp(X, tree_clf).plot(kind='barh', figsize=(12,7), legend=False)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fd1d80fbb50>



CONCLUSION

In this project we implemented random forest classification, logistic regression, XGboost classifier, K-nearest neighbours, SVM, decision tree classification and among all of this we got highest accuracy from logistic regression model which is 83.61%.

CONSOLIDATION:

Machine Learning Models	**Accuracy scores**
Random Forest Classifier	77.05%
Logistic Regression	83.67%
KNN	62.30%
SVM	80.33%
XG Boost	75.41%
Decision Tree	75.40%

RESULTS AND CONCLUSION:

This project provides the deep insight into machine learning techniques for classification of heart diseases. The role of classifier is crucial in healthcare industry so that the results can be used for predicting the treatment which can be provided to patients. The existing techniques are studied and compared for finding the efficient and accurate systems.

With the increasing number of deaths due to heart diseases, it has become mandatory to develop a system to predict heart diseases effectively and accurately. The motivation for the study was to find the most efficient ML algorithm for detection of heart diseases. We here compare the accuracy score of Random Forest, Logistic Regression, KNN, SVM, XG Boost Classifier and Decision tree algorithms for predicting heart disease using UCI machine learning repository dataset.

The result states that the Logistic regression model is the most efficient with accuracy score of 83.67% for prediction of heart disease. In future the work can be enhanced by developing a web application based on the algorithm.

Machine learning techniques significantly improves accuracy of cardiovascular risk prediction through which patients can be identified during an early stage of disease and can be benefitted by preventive treatment. I conclude with a statement that there is a huge scope for machine learning algorithms in predicting cardiovascular diseases or heart related diseases.

REFERENCES:

- UCI, Heart Disease Data Set.[Online]. Available (Accessed on May 1 2020): https://www.kaggle.com/ronitf/heart-disease-uci.
- Avinash Golande, Pavan Kumar T, Heart Disease Prediction Using Effective Machine Learning Techniques, International Journal of Recent Technology and Engineering, Vol 8, pp.944-950,2019.
- T.Nagamani, S.Logeswari, B.Gomathy, Heart Disease Prediction using Data Mining with Mapreduce Algorithm, International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-8 Issue-3, January 2019.
- Internet source [Online].Available (Accessed on May 1 2020): http://acadpubl.eu/ap
- T. Porter and B. Green, "Identifying Diabetic Patients: A Data Mining Approach," Americas Conference on Information Systems, 2009
- Ramalingam VV, Dandapath A, Raja MK. Heart disease prediction using machine learning techniques: a survey. Int J Eng Technol. 2018;7(2.8):684–7.
- Vembandasamy K, Sasipriya R, Deepa E. Heart diseases detection using Naive Bayes algorithm. Int J Innov Sci Eng Technol. 2015;2(9):441–4.
- Parthiban G, Srivatsa SK. Applying machine learning methods in diagnosing heart disease for diabetic patients. Int J Appl Inf Syst (IJAIS). 2012;3(7):25–30.
- Chaurasia V, Pal S. Data mining approach to detect heart diseases. Int J Adv Comput Sci Inf Technol (IJACSIT). 2014;2:56–66.
- Weng SF, Reps J, Kai J, Garibaldi JM, Qureshi N. Can machine-learning improve cardiovascular risk prediction using routine clinical data? PLoS ONE. 2017;12(4):e0174944.