Heart Failure Prediction

Author: Nizar Assad

Year: 2021-2022

Table of Contents

Abstract

Heart-related diseases are the main reason for a huge number of death in the world over the last decades and is considered as the most life-threatening disease. Our project object is to detect if a person has a heart disease or not. The main goal of this research is to improve the accuracy of diagnosis and help saving human resources. For this we will conduct a research to improve heart failure prediction using the 'heart.csv' dataset.

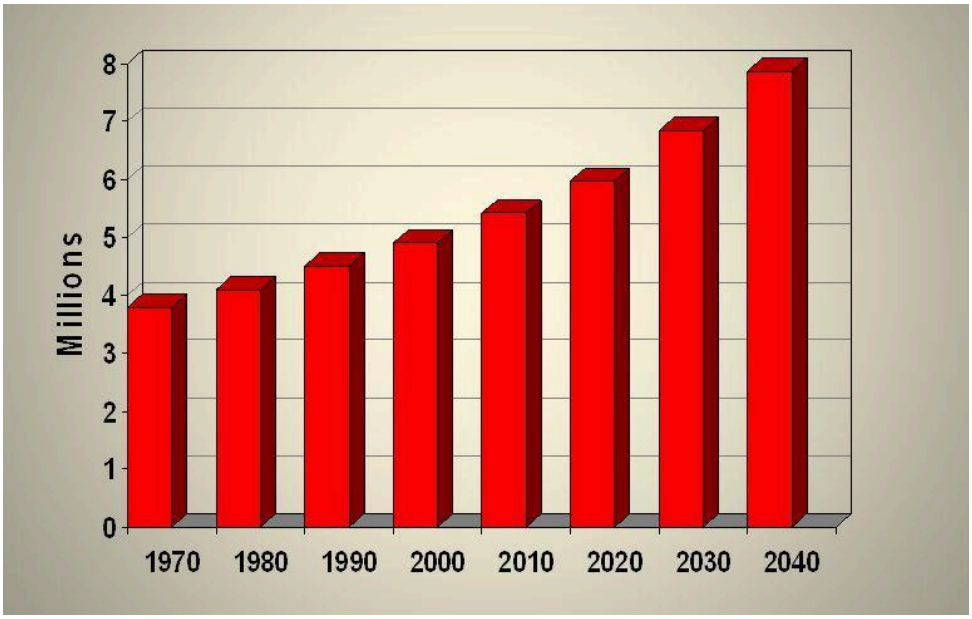
K*eywords: Machine Learning, Logistic regression, Decision tree, SVM, Heart disease.*

Introduction

A heart failure, also known as congestive heart failure, happens when the heart doesn't pump the amount of blood the body requires, which causes shortness of breath. Our problem is that we want to predict whether a person has a heart disease or not. According to the world-wide Organization, 17,9 million people die each year due to heart disease. Many deaths in all countries of the world are due to heart ailments. (Cardiovascular diseases, 2021)
To succeed in our task, the proposed model should be at the same time efficient and reliable, because the consequence are big in case the prediction is false. If a person with no heart disease is predicted with heart disease, it will have a serious impact on the patient's mental health a cause him panic for no reason. Otherwise, if the patient has a heart disease and is predicted to be with no heart disease, he will miss the best and maybe the last chance to cure his disease. For both cases, nobody can afford any wrong diagnosis as it is very painful. If the accuracy is high enough we might encounter a large number of issues and solve all the unnecessary trouble. So, if the model is reliable, it can be applied in medical prediction.

The graph below shows the fast growth of heart failure patients from 1970 – 2040:

Heart Failure demographics



(Evander Holyfield, 2021)

The main goal of this data exploration and prediction is to have more insight on the health factors that causes a patient's risk for heart disease. For this, we will explore the data,

apply some Python code implementation and then start the data prediction using three different classification models that we will compare using accuracy.

Background

In our century, people care more about their bank accounts and their net worth more than they care about their own healthcare which led to a high increase of mental health issues and physical stress. Also, a huge impact of the corona crisis which apparently causes heart suffer with muscle damage and affects heart function. (Susan Post, M.D., M.S., 2021)

Many health centers and medical organizations collect a huge amount of data about heart issues. With all this information in hand, we can apply various machine learning techniques to exploit the data and have a closer perception.

We conducted research on how people investigated heart disease prediction before conducting the studies so that we might widen our perspectives and learn from them.

Ujma Ansari used the Decision Tree model to predict heart illness in 2011 and achieved a high accuracy of 99 percent, inspiring us to utilize a superior version of Decision Tree, Random Forest. (Soni, Jyoti, 2011)

Chaitrali S. Dangare produced the forecast in 2012 using three different models, including Nave Bayes, Decision Trees, and Neural Networks. We're working with the identical data set as he did. The difference between his work and ours is that he added two more characteristics to the dataset, bringing the total number of features to 15, compared to 13 in ours. Despite the fact that there is no significant difference between 13 and 15 features in his work, what he accomplished with the dataset motivates us to make a meaningful improvement to our dataset to avoid overfitting. (Dangare, Chaitrali S., and Sulabha S. Apte, 2012)

Aim

The aim of this project is to predicts the likelihood of patients getting heart disease, providing more knowledge. Therefore, allowing researchers to develop better ways to prevent this from happening and establish better patterns.

Objectives

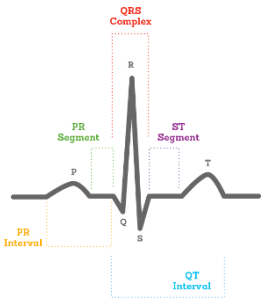
- To develop machine learning models, helping to predict possibilities of heart disease. For this, we will implement Logistic regression, Decision tree, SVM, and more classification models.
- To detect the high-risk factors which may cause a heart issue.
- To analyze different classification models applied on the 'heart.csv' and compare their accuracy scores.

Heart failure prediction dataset

The Heart Failure prediction dataset selected for this project comes from the Kaggle Repository. The dataset consists of 918 patients' data. Describing the individual's diagnosis of heart disease. After a long research, this dataset was chosen for its results in this particular field, it offers a good compromise between efficiency and simplicity.

This data is mainly focused on analyzing the heart condition to predict the binary target, if a person is at high risk of having a heart disease or not.

All this data is based on the ECG curves which represents the functioning of a person's heart by a cardiac rhythm.



The 12 features which represents health factors used in the project are listed below:

- Age: age in years
- Sex: (1 = male; 0 = female)
- ChestPainType: [TA, ATA, NAP, ASY].
- RestingBP: resting blood pressure (in mm Hg on admission to the hospital)

- Cholesterol: serum cholesterol in mg/dl
- FastingBS: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- RestingECG: resting electrocardiographic results
- MaxHR: maximum heart rate achieved
- ExerciseAngina: exercise induced angina (1 = yes; 0 = no)
- Oldpeak: ST depression induced by exercise relative to rest
- ST_Slope: The slope of the peak exercise ST segment
- HeartDiceaser: Output column [1: Heart disease, 0: Normal]

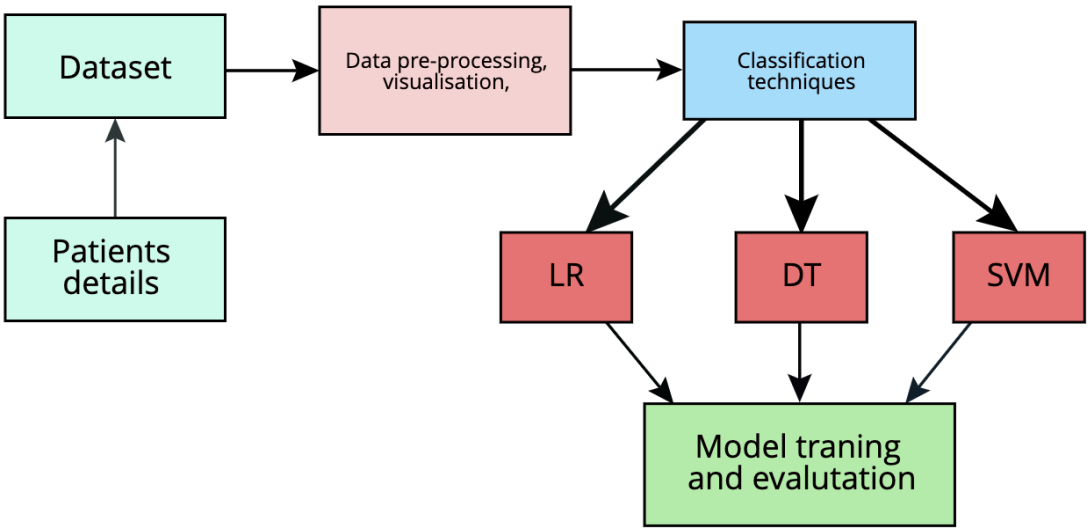
Problem to be addressed

The major problem with heart disease is to detect it. There are many services available to check your heart and do a full diagnosis but the prices are usually too high and not 100% reliable. Also, if the disease is detected early can be very helpful as it can aid in making decisions on lifestyle changes in high risk patients which will certainty reduce the risk of a heart attack. In today's world, we have enough data to analyze it and try to spot the most relevant risk factors and be in help fighting this condition.

Machine learning model

1- Summary of the approach

This diagram shows the steps we're following in order to proceed with the dataset prediction.



Data pre-processing

Libraries needed in the study

- Obtaining data is the initial stage in the machine learning process. We employ a variety of libraries to accomplish our objectives, including Panda for data manipulation and analysis, NumPy for array operations, Matplotlib for producing aesthetically appealing and educational statistics graphics, and Sci-kit for statistical modelling. These are the librares that Data Scientists utilise the most. (I used Google Colab Notebook for this.)

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import plotly.express as px
import plotly.figure_factory as ff
import plotly.graph_objects as go
from plotly.offline import iplot
from plotly import tools
import seaborn as sns
from sklearn.preprocessing import StandardScaler, PolynomialFeatures, OneHotEncoder, StandardScaler, PowerTransformer, MinMaxScaler, LabelEncoder, Ro
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import plot_confusion_matrix, r2_score, mean_absolute_error, mean_squared_error, classification_report, confusion_matrix, accura
from sklearn.metrics import plot_roc_curve, roc_auc_score, roc_curve, f1_score, accuracy_score, recall_score
from sklearn import tree
import sys
np.set_printoptions(threshold=sys.maxsize)
from sklearn.compose import ColumnTransformer
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
```

Importing the data

- We imported the ‘heart.csv’ dataset and name it ‘data’ using the ‘ pd ’ notation.

```
#we named our dataframe 'data'
data = pd.read_csv('heart.csv')
```

General looking on data

- Using the head and tail functions to output the top and last 5 rows in the data set (set by default) for a quick look on data frame to get a brief idea.

```
#quick look on datafram
data.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

```
data.tail()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	0

- We've assigned EDA as a copy of data. Using the shape property to output the current shape of the array.

```
EDA = data.copy()
EDA.shape
```

```
(918, 12)
```

- The info() function is used to print a concise summary of a DataFrame. From this summary we can have a lot of information of data.

- Range Index.
- Number of columns.
- If there's any null values in any of the columns.
- Type of data of every feature.
- Memory usage.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

- We will use the describe function to get a summary statistic of the numerical features within the data frame. For this example, we separate the data in two different tables, one for the target value '0' and a second one for the '1' values.

['HeartDisease'] == 0

```
data[data['HeartDisease']==0].describe().T.style.background_gradient(subset=['mean','std','50%','count'], cmap='RdPu')
```

	count	mean	std	min	25%	50%	75%	max
Age	410.000000	50.551220	9.444915	28.000000	43.000000	51.000000	57.000000	76.000000
RestingBP	410.000000	130.180488	16.499585	80.000000	120.000000	130.000000	140.000000	190.000000
Cholesterol	410.000000	227.121951	74.634659	0.000000	197.250000	227.000000	266.750000	564.000000
FastingBS	410.000000	0.107317	0.309894	0.000000	0.000000	0.000000	0.000000	1.000000
MaxHR	410.000000	148.151220	23.288067	69.000000	134.000000	150.000000	165.000000	202.000000
Oldpeak	410.000000	0.408049	0.699709	-1.100000	0.000000	0.000000	0.600000	4.200000
HeartDisease	410.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

['HeartDisease'] == 1

```
data[data['HeartDisease']==1].describe().T.style.background_gradient(subset=['mean','std','50%','count'], cmap='YlGn')
```

	count	mean	std	min	25%	50%	75%	max
Age	508.000000	55.899606	8.727056	31.000000	51.000000	57.000000	62.000000	77.000000
RestingBP	508.000000	134.185039	19.828685	0.000000	120.000000	132.000000	145.000000	200.000000
Cholesterol	508.000000	175.940945	126.391398	0.000000	0.000000	217.000000	267.000000	603.000000
FastingBS	508.000000	0.334646	0.472332	0.000000	0.000000	0.000000	1.000000	1.000000
MaxHR	508.000000	127.655512	23.386923	60.000000	112.000000	126.000000	144.250000	195.000000
Oldpeak	508.000000	1.274213	1.151872	-2.600000	0.000000	1.200000	2.000000	6.200000
HeartDisease	508.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Notice the change in the color in the two tables by inputting different value to 'cmap'

Data visualization

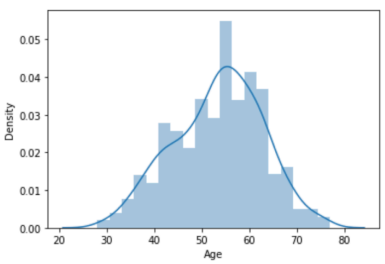
Numerical features:

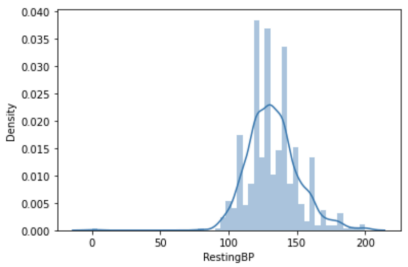
To visualize the data, we started with the numerical features. For this, we created a list names ‘num_feat’ with all the corresponding features.

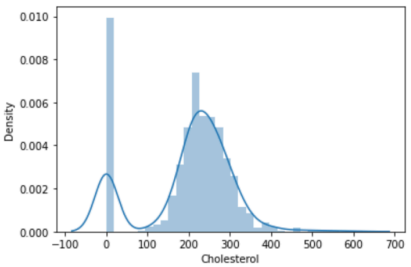
```
num_feat = ["Age",
            "RestingBP",
            "Cholesterol",
            "FastingBS",
            "MaxHR",
            "Oldpeak",
            "HeartDisease"
]
```

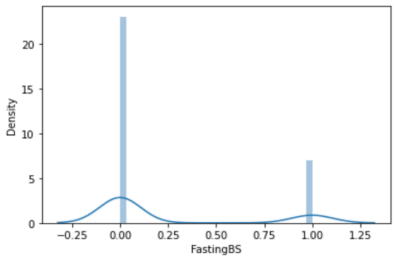
- Visualize the data with distplots using the matplotlib and seaborn libraries.

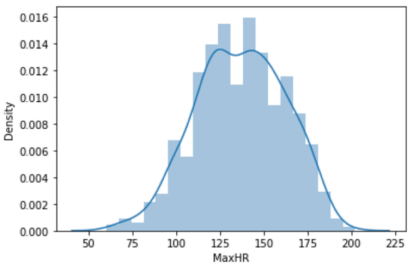
```
for col in EDA[num_feat]:
    plt.figure()
    sns.distplot(EDA[col])
    plt.show()
```

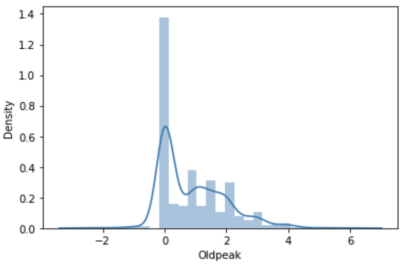


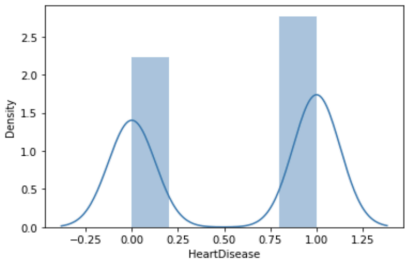








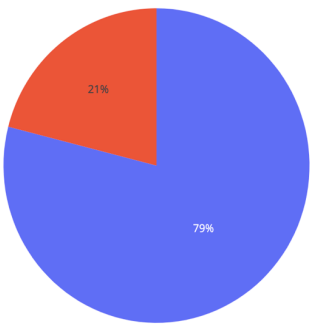


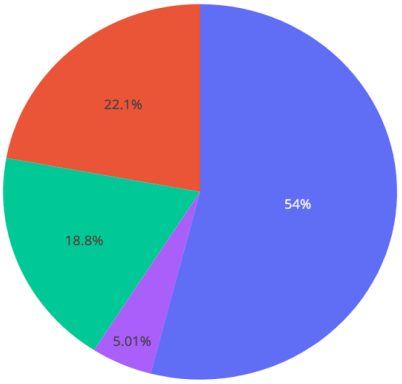


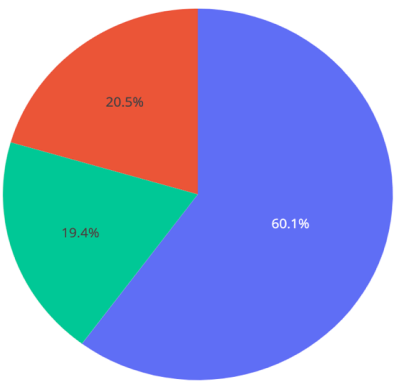
Categorical features:

- Circle graphs seem to be the best fit for visualizing the categorical data, we used the 'graph_objects' provided with the matplotlib library.

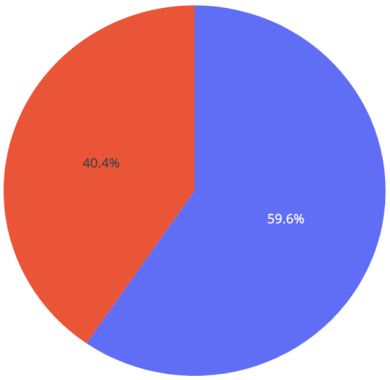
```
fig = go.Figure(data=[go.Pie(labels=data['Sex'].value_counts().index,
                             values=data['Sex'].value_counts().values)])

fig.update_layout(
    title_text="Sex ")
fig.show()
```

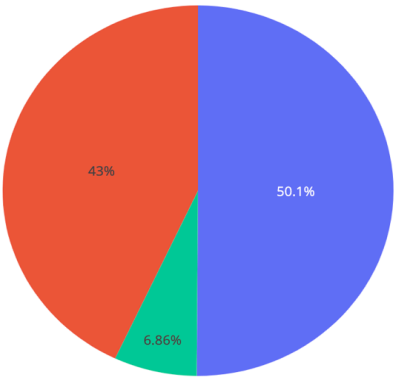


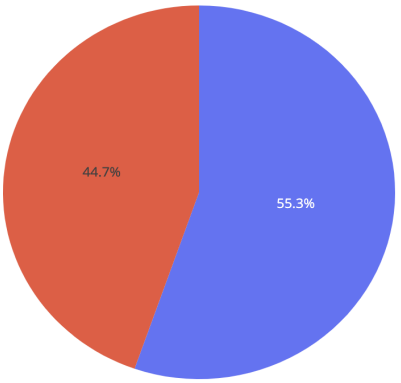




Sex **Chest pain type.** **RestingECG**







ExerciseAngina **ST_Slop** **Heart disease**
Target

Notice: The target values (heart disease feature) are balanced.

Cleaning the data:

- Any machine learning algorithm's performance and accuracy suffer as a result of these null values. As a result, it's critical to eliminate null values from a dataset before using a machine learning algorithm on it.

```
data.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
```

There’s no missing data as we can see in the output above.

- Checking for duplicated rows is crucial for the cleaning process as it can affect the accuracy and it makes the dataset weaker and less efficient.

```
data.duplicated().sum()
```

```
0
```

There's no duplicated rows in our dataset.

Relationship analysis:

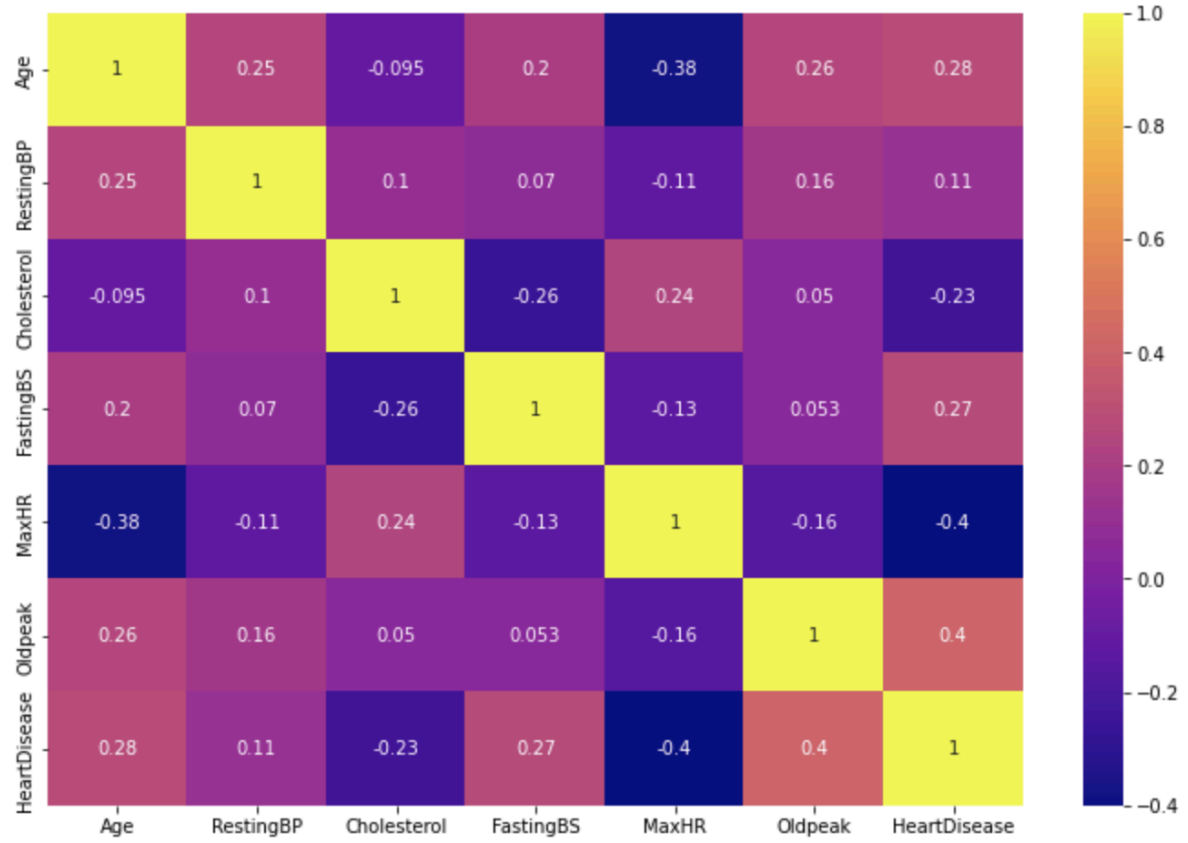
Data correlation

To check the data correlation is so important because if two variables have a linear connection, correlation can tell you how strong that relationship is.

- The heatmap is used to represent correlation between features, comes with the Seaborn python library.

```
plt.figure(figsize=(12,8))
sns.heatmap(data.corr(),cmap='plasma',annot=True)
```

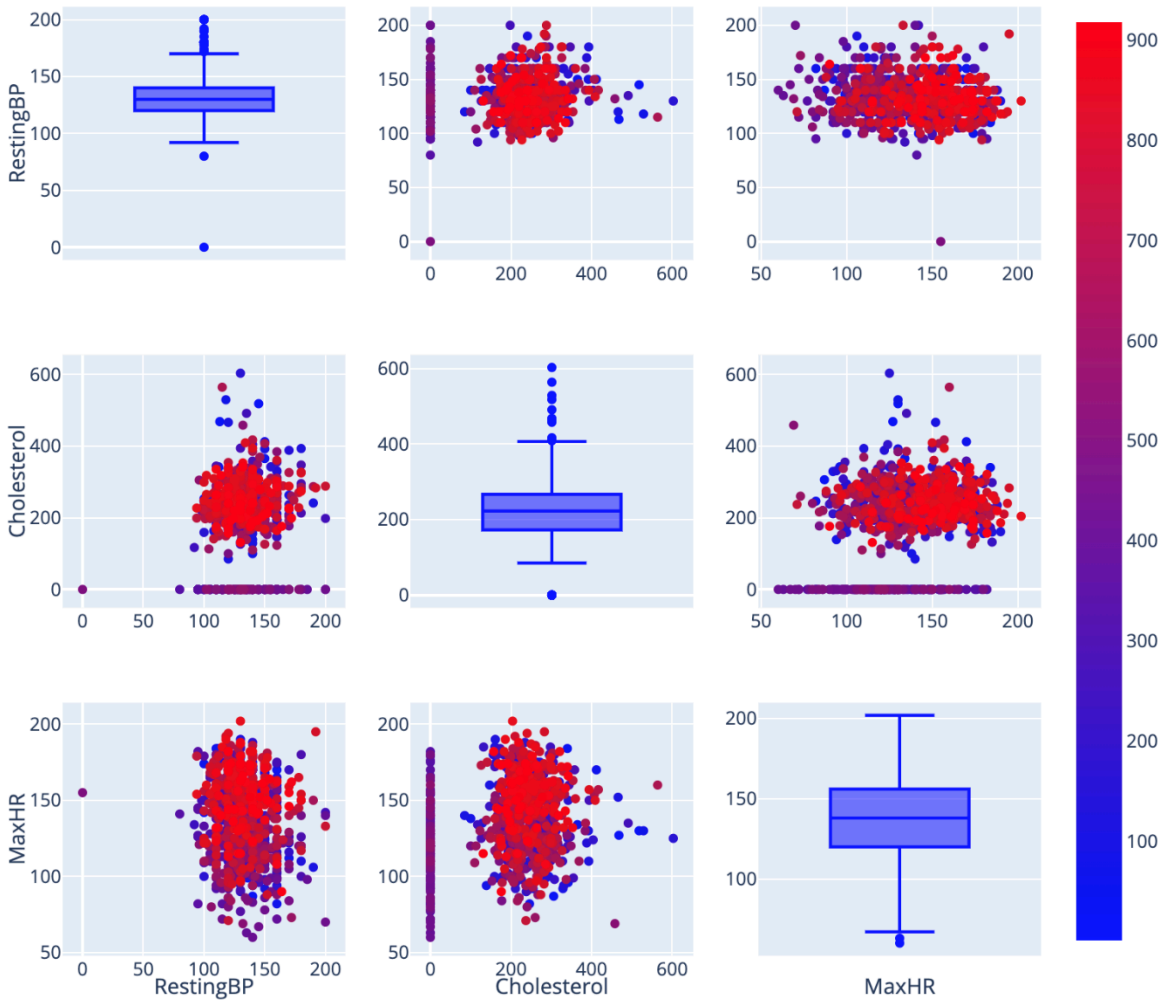
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f74d5b110>
```



Notice: As we see from the heatmap, all features have a correlation to heart diseases. So, all the attributes will impact the prediction.

- Using the scatterplot matrix to visualize the correlation between the selected features. (we've selected these features because they have a high number of unique values)

```
data_forplot = data.loc[:,['RestingBP', 'Cholesterol', 'MaxHR']]
data_forplot["index"] = np.arange(1,len(data_forplot)+1)
fig = ff.create_scatterplotmatrix(data_forplot, diag='box', index='index',colormap='Bluered',
                                  height=800, width=800)
iplot(fig)
```



Notice: As we can see in the scatterplotmatrix, there are some outliers than needs to be removed. But after a long analysis in data, I think that all the values will be needed for this prediction as it covers all the types of patients that we could be asked to predict using the right machine learning algorithm.

Splitting the target and the features

- We isolate the source variables (independent variables) from the target variable to be predicted (dependent variable) in order to simplify the next steps of data pre-processing, by adding these lines to ease the next phases of data preprocessing:

```
x = data.iloc[:,:-1].values
y = data.iloc[:,-1].values
```

Here we called the ‘x’ array to see if the target feature separation was done successfully. And we can that this only contain 11 columns which confirms it.

```
x
```

```
array([[40, 'M', 'ATA', 140, 289, 0, 'Normal', 172, 'N', 0.0, 'Up'],
       [49, 'F', 'NAP', 160, 180, 0, 'Normal', 156, 'N', 1.0, 'Flat'],
       [37, 'M', 'ATA', 130, 283, 0, 'ST', 98, 'N', 0.0, 'Up'],
       [48, 'F', 'ASY', 138, 214, 0, 'Normal', 108, 'Y', 1.5, 'Flat'],
       [54, 'M', 'NAP', 150, 195, 0, 'Normal', 122, 'N', 0.0, 'Up'],
       [39, 'M', 'NAP', 120, 339, 0, 'Normal', 170, 'N', 0.0, 'Up'],
       [45, 'F', 'ATA', 130, 237, 0, 'Normal', 170, 'N', 0.0, 'Up'],
```

```
[22] x.shape
```

```
(918, 11)
```

And the ‘y’ array contains one column values which represents the target feature

```
[23] y.shape
```

```
(918,)
```

- Because machine learning models are built on mathematical equations, it's reasonable to think that keeping categorical data in the equations would cause issues because we only want numbers in the equations. That’s why we need to use the LabelEncoder function which transforms categorical features to numbers.

```
#Label Encoding the object dtypes.
le = LabelEncoder()
x[:,1] = le.fit_transform(x[:,1])
x[:,6] = le.fit_transform(x[:,6])
x[:,8] = le.fit_transform(x[:,8])
```

- Now let’s call the ‘x’ array to check if the encoding has succeeded.

```
x
```

```
array([[40, 1, 'ATA', 140, 289, 0, 1, 172, 0, 0.0, 'Up'],
       [49, 0, 'NAP', 160, 180, 0, 1, 156, 0, 1.0, 'Flat'],
       [37, 1, 'ATA', 130, 283, 0, 2, 98, 0, 0.0, 'Up'],
       [48, 0, 'ASY', 138, 214, 0, 1, 108, 1, 1.5, 'Flat'],
       [54, 1, 'NAP', 150, 195, 0, 1, 122, 0, 0.0, 'Up'],
       [39, 1, 'NAP', 120, 339, 0, 1, 170, 0, 0.0, 'Up'],
       [45, 0, 'ATA', 130, 237, 0, 1, 170, 0, 0.0, 'Up'],
       [54, 1, 'ATA', 110, 208, 0, 1, 142, 0, 0.0, 'Up'],
       [37, 1, 'ASY', 140, 207, 0, 1, 130, 1, 1.5, 'Flat'],
       [48, 0, 'ATA', 120, 284, 0, 1, 120, 0, 0.0, 'Up'],
       [37, 0, 'NAP', 130, 211, 0, 1, 142, 0, 0.0, 'Up'],
       [58, 1, 'ATA', 136, 164, 0, 2, 99, 1, 2.0, 'Flat'],
       [39, 1, 'ATA', 120, 204, 0, 1, 145, 0, 0.0, 'Up'],
       [49, 1, 'ASY', 140, 234, 0, 1, 140, 1, 1.0, 'Flat'],
       [42, 0, 'NAP', 115, 211, 0, 2, 137, 0, 0.0, 'Up'],
```

Notice: As we can see in the output array above, all the called features have been transformed to numerical data using the 'fit_transform' function.

- Another typical approach for dealing with categorical information is one-hot encoding. It simply adds more characteristics to the category feature dependent on the number of unique values. Every category's unique value will be included as a feature.

```
enc = OneHotEncoder()
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [2 ,10])], remainder='passthrough')
x = np.array(ct.fit_transform(x))
```

Here's the output:

```
x
```

```
array([[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 40, 1, 140, 289, 0, 1, 172, 0,
        0.0],
       [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 49, 0, 160, 180, 0, 1, 156, 0,
        1.0],
       [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 37, 1, 130, 283, 0, 2, 98, 0,
        0.0],
       [1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 48, 0, 138, 214, 0, 1, 108, 1,
        1.5],
       [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 54, 1, 150, 195, 0, 1, 122, 0,
        0.0],
       [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 39, 1, 120, 339, 0, 1, 170, 0,
        0.0],
       [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 45, 0, 130, 237, 0, 1, 170, 0,
```

As we can see in the output above, ChestPain type and ST_Slope features have been successfully encoded and separated in features.

Splitting the data into training & test data

After we make sure that these three points are accomplished:

- Data is clean
- Target and features are splitted.
- All features are numerical

We can start splitting data into train and test set to train our models using our dataset and then test it.

- Using the 'train_test_split' functions imported from the sklearn library. We've assigned 20% of the data to the test section as we have specified test_size = 0.2. And the rest will be automatically taken by the training section.

```
x_train, x_test, y_train, y_test= train_test_split(x,y, test_size=0.2 , stratify=y, random_state=1)
```

- x_train: contains the values of the features
- y_train: contains the target output corresponding to x_train values
- x_test: contains the values of the features to be tested after training
- y_test: contains category labels for your test data, which will be used to compare actual and projected categories.

Feature scaling:

The preprocessing step of feature scaling through standardization is critical for the machine learning algorithms we're applying. Rescaling the characteristics such that they have the attributes of a conventional normal distribution with a mean of zero and a standard deviation of one is known as standardization.
`

- For this, we use the 'MinMaxScaler' function after importing it from the sklearn library.

```
S = MinMaxScaler()
```

- We've assigned two variable (x_train_scaled, x_test_scaled) that will store the scaled values for the x_train and x_test arrays.

```
x_train = S.fit_transform(x_train)
x_test = S.fit_transform(x_test)
```

Now, let's call 'x_train' and see if the scaling has succeeded:

```
[30] x_train

     array([[1.        , 0.        , 0.        , 0.        , 0.        ,
             1.        , 0.        , 0.53061224, 1.        , 0.625     ,
             0.35820896, 0.        , 0.5       , 0.56338028, 0.        ,
             0.24390244],
            [1.        , 0.        , 0.        , 0.        , 0.        ,
             0.        , 1.        , 0.69387755, 0.        , 0.62      ,
             0.34660033, 0.        , 0.5       , 0.72535211, 0.        ,
             0.24390244],
            [1.        , 0.        , 0.        , 0.        , 0.        ,
             1.        , 0.        , 0.73469388, 1.        , 0.72      ,
             0.        , 0.        , 1.        , 0.43661972, 1.        ,
             0.36585366],
            [0.        , 0.        , 0.        , 1.        , 0.        ,
             0.        , 1.        , 0.08163265, 1.        , 0.475     ,
             0.        , 1.        , 0.5       , 0.47183099, 0.        ,
             0.32926829],
            [1.        , 0.        , 0.        , 0.        , 0.        ,
```

As we can see from the output above, the scaling has been applied to the array.

Model training and evaluation

1- Logistic regression:

The first model we decide to use is Logistic regression, it is a type of supervised learning in which the probability for classification problems with two outcomes are computed. It may also be used to predict many classes like in our example. LR_model have been known for its high accuracy rates in prediction datasets and also for Its efficiency.

Loading and training the model:

- Here, we load the logistic regression into the variable that we called 'LR'

```
#Load Logistic regression model to the 'LR' variable
LR = LogisticRegression()
```

After the model have been created, we need to fit in our training data in order to train the model.

- For this, we will use the fit method.

```
#training the model using train data
LR.fit(x_train, y_train)
```

Testing the model:

After training our model, we need to test it using the testing data we've assigned before. We need to use the scaled features in order to get a good accuracy score.

- For this, we used the predict() function, it allows us to predict the labels of data values on the basis of the trained model.

```
y_pred_LR = LR.predict(x_test)
```

2- Decision trees:

Because they are both simple to learn and effective, decision trees are often the tool of choice for predictive modelling. A decision tree's main purpose is to divide a large amount of data into smaller segments.

Loading and training the model:

- Here, we load the Decision tree model into the variable we called 'DT'

```
#Load Decision tree model to the 'DT' variable
DT = tree.DecisionTreeClassifier()
```

After the model have been created, we need to fit in our training data in order to train the model.

- For this, we will use the fit method.

```
#training the model using train data
DT.fit(x_train, y_train)
```

```
DecisionTreeClassifier()
```

Testing the model:

Let's test the DT model.
Using the predict function that will predict the target value based on the values captured from the 'x_test' array.

```
y_pred_DT = DT.predict(x_test)
```

3- Support Vector Machine (SVM):

SVMs (support vector machines) are a collection of supervised learning algorithms for categorization. The efficacy of support vector machines in high-dimensional areas is one of its advantages.

Loading and training the model:

- Here, we load the SVM model into the variable we called 'clf'.

```
#Load SVM tree model to the 'svm' variable
clf = svm.SVC()
```

After the model have been created, we need to fit in our training data in order to train the model.

- For this, we will use the fit method.

```
#training the model using train data
clf.fit(x_train, y_train)
```

```
SVC()
```

Testing the model:

Let's test the 'clf' model.
Using the predict function that will predict the target value based on the values captured from the 'x_test' array.

```
y_pred_clf = clf.predict(x_test)
```

4- Random forest

Random forest is a versatile, easy-to-use machine learning technique that, in most cases, gives excellent results even without hyper-parameter tuning.

Loading and training the model:

- Loading the Random forest model into the variable we called 'RF'.

```
#Load Random forest tree model to the 'RF' variable
RF = RandomForestClassifier()
```

After the model have been created, we need to fit in our training data in order to train the model.

- Fit the data in the RF model, we will use the fit method.

```
#training the model using train data
RF.fit(x_train, y_train)
```

Testing the model:

- Let's test the 'RF' model.

```
y_pred_RF = RF.predict(x_test)
```

5- K-Nearest neighbors

the k-nearest neighbors algorithm (k-NN) is a non-parametric classification method.

Loading and training the model:

- Loading the model into the variable we called 'nbrs'

```
#Load K-Nearest Neighbours model to the 'nbrs' variable
nbrs = KNeighborsClassifier()
```

After the model have been created, we need to fit in our training data in order to train the model.

- Fitting our training data.

```
#training the model using train data
nbrs.fit(x_train, y_train)
```

```
KNeighborsClassifier()
```

Testing the model:

```
y_pred_nbrs = nbrs.predict(x_test)
```

6- Gaussian Naïve Bayes

Gaussian Naive Bayes is a Naive Bayes variation that allows continuous data and follows the Gaussian normal distribution.

Loading and training the model:

- Loading the Gaussian Naive Bayes model into the variable called 'gnb'.

```
#Load Gaussian Naive Bayes model to the 'gnb' variable
gnb = GaussianNB()
```

- Fitting data.

```
#training the model using train data
gnb.fit(x_train, y_train)
```

Testing the model:

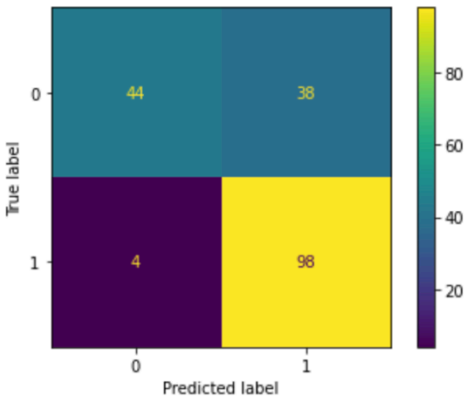
- Testing the 'gnb' model.

```
y_pred_gnb = gnb.predict(x_test)
```

Evaluating the models:

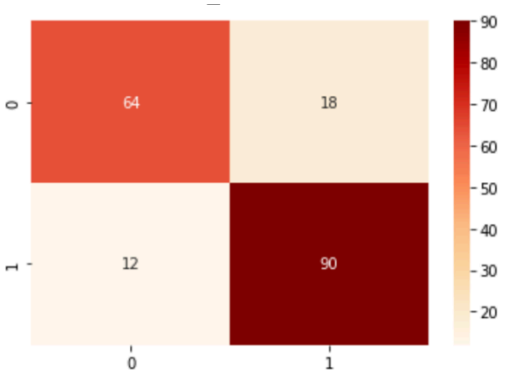
It's essential to evaluate our models as we're building it to find the most accurate predictions I utilized the data without applying the feature scaling to train and test the models in my initial effort to evaluate the methods functioning on this project. The findings were disappointing, therefore I employed feature scaling, which resulted in a significant change in evaluation outcomes.
The table below show different classification metrics and confusion matrix applied to the logistic regression model before and after using feature scaling.

Before applying FS (LR)	After applying FS (LR)



	train	test
Accuracy	0.787466	0.771739
Precision	0.727273	0.720588
Recall	0.985222	0.960784
f1	0.836820	0.823529

Accuracy: 78%



	precision	recall	f1-score	support
0	0.84	0.78	0.81	82
1	0.83	0.88	0.86	102
accuracy			0.84	184
macro avg	0.84	0.83	0.83	184
weighted avg	0.84	0.84	0.84	184

Accuracy: 84%

- To evaluate and validate the models, we used accuracy, precision, recall, F1 score, and cross validation. We picked these parameters since they are the best match for our project, which is about classification.

```
cm = confusion_matrix(y_test, y_pred_LR)
LR_report = classification_report(y_test, y_pred_LR)
f1_score(y_test ,y_pred_LR)
accuracy_score(y_test ,y_pred_LR)
```

Logistic Regression

Acc: 83%

```
cm = confusion_matrix(y_test, y_pred_DT)
DT_report = classification_report(y_test, y_pred_DT)
f1_score(y_test ,y_pred_DT)
accuracy_score(y_test ,y_pred_DT)
```

Decision tree

Acc: 66%

```
cm = confusion_matrix(y_test, y_pred_clf)
clf_report = classification_report(y_test, y_pred_clf)
f1_score(y_test ,y_pred_clf)
accuracy_score(y_test ,y_pred_clf)
```

SVM

Acc: 84%

```
cm = confusion_matrix(y_test, y_pred_RF)
RF_report = classification_report(y_test, y_pred_RF)
f1_score(y_test ,y_pred_RF)
accuracy_score(y_test ,y_pred_RF)
```

Random Forest

Acc: 83%

```
cm = confusion_matrix(y_test, y_pred_nbrs)
nbrs_report = classification_report(y_test, y_pred_nbrs)
f1_score(y_test ,y_pred_nbrs)
accuracy_score(y_test ,y_pred_nbrs)
```

K nearest neighbors

Acc: 86%

```
cm = confusion_matrix(y_test, y_pred_gnb)
gnb_report = classification_report(y_test, y_pred_gnb)
f1_score(y_test ,y_pred_gnb)
accuracy_score(y_test ,y_pred_gnb)
```

Gaussian Naïve Bays

Acc: 84%

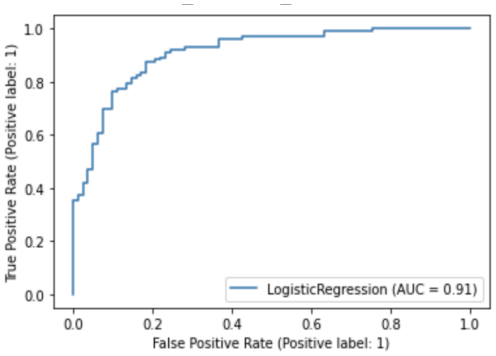
The K nearest neighbors has the best accuracy score of 86%.

- We included all the results outputted from the sections of code above in one table for the 6 Classification methods used during this research.

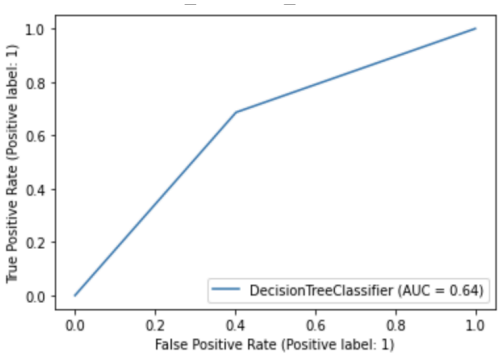
	Precision		recall		f1-score		Cross-Val	
	Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**Acc%**	**SD**
Logistic Regression	0.84	0.83	0.78	0.88	0.81	0.86	84	0.04
Decision Tree	0.62	0.70	0.65	0.68	0.63	0.69	75	0.05
SVM	0.86	0.82	0.76	0.90	0.81	0.86	83	0.04
Random Forest	0.89	0.80	0.71	0.93	0.79	0.86	83	0.04
K-Nearest Neighbors	0.88	0.85	0.79	0.91	0.83	0.88	66	0.05
Gaussian Naive Bayes	0.88	0.79	0.70	0.92	0.78	0.85	0.83	0.04

ROC curves:

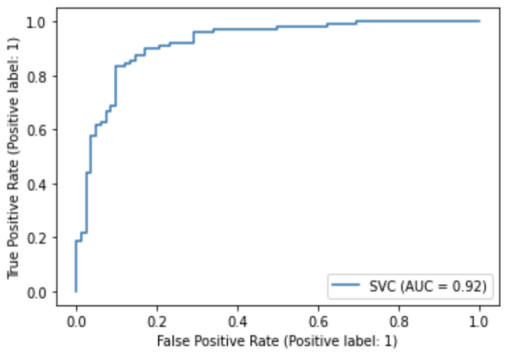
- We used ROC curves to see the performance of the trained and tested classification models at all classification thresholds.



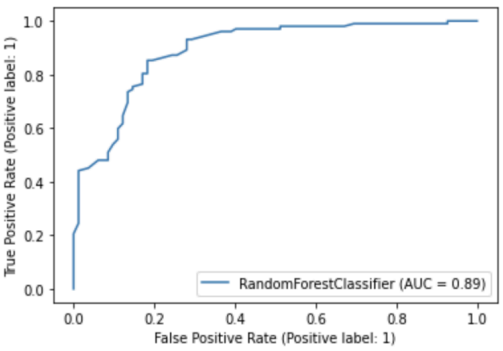
LogisticRegression

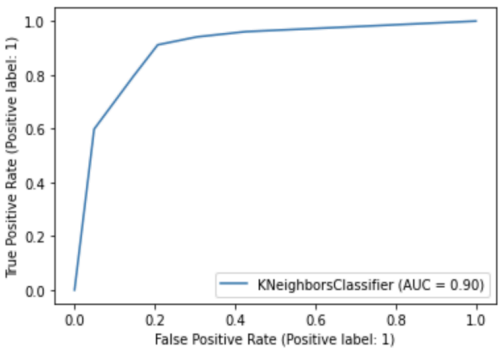


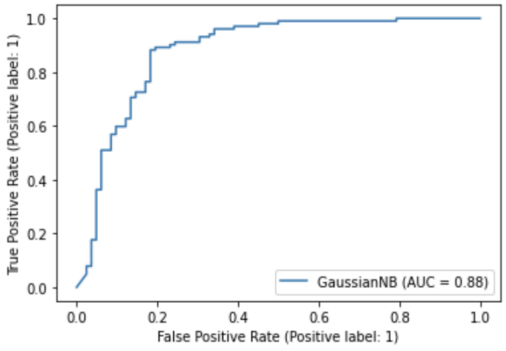
DecisionTreeClassifier



SVM







Random Forest K-Nearest Neighbors Gaussian Naive Bayes

As we can see in the graphs above, we can directly say that the K-Nearest Neighbors model offers the best compromise in terms of regularization.

Confusion matrix:

We will use confusion matrix to describe the performance of the used classification models, It allows the visualization of the performance of an algorithm and easy identification of confusion between classes.
Four types of outputs can be expected when performing the classification prediction.

We'd want to go through the definitions of TP, FP, TN, and FN.
Four types of outputs can be expected when performing the classification prediction:

- True positive (TP)= When you predict that an observation belongs to a class, and it turns out that it does.
- False Positive (FP)= When you predict that an observation does not belong to a class, and it does not belong to that class.
- True Negative (TN)= arise when you incorrectly guess that an observation belongs to a class when it does not.
- False Negative (FN) = occur when you incorrectly guess that an observation does not belong to a class when it does.

(Evaluating a machine learning model., 2021)

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

(Draelos, 2019)

- For this, we'll use the heatmap function imported from the seaborn library.

- Logistic Regression (LR):

```
sns.heatmap(confusion_matrix(y_test, y_pred_LR), annot=True, cmap='OrRd', fmt='g')
```

- Decision Tree (DT):

```
sns.heatmap(confusion_matrix(y_test, y_pred_DT), annot=True, cmap='OrRd', fmt='g')
```

- SVM (clf):

```
sns.heatmap(confusion_matrix(y_test, y_pred_clf), annot=True, cmap='OrRd', fmt='g')
```

- Rabdom forest (RF):

```
sns.heatmap(confusion_matrix(y_test, y_pred_RF), annot=True, cmap='OrRd', fmt='g')
```

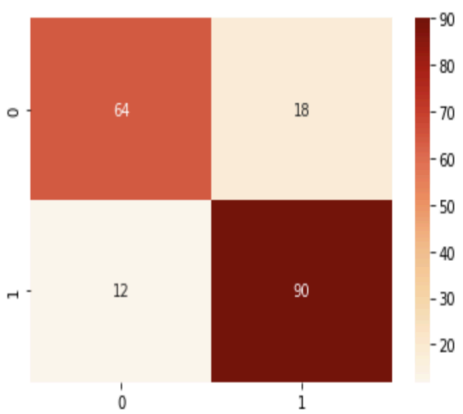
- K-nearest neighbors (nbrs):

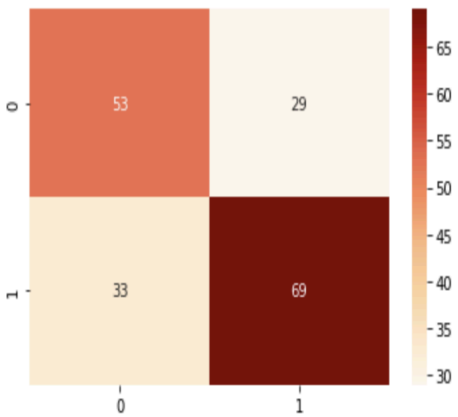
```
sns.heatmap(confusion_matrix(y_test, y_pred_nbrs), annot=True, cmap='OrRd', fmt='g')
```

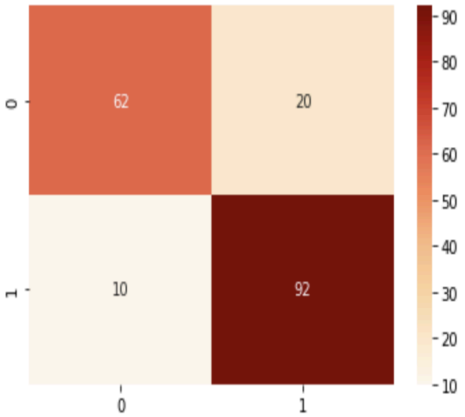
- Gaussian naïve bayes (gnb):

```
sns.heatmap(confusion_matrix(y_test, y_pred_gnb), annot=True, cmap='OrRd', fmt='g')
```

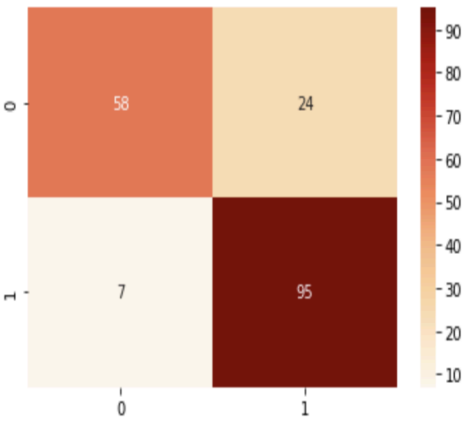
Here's the output:



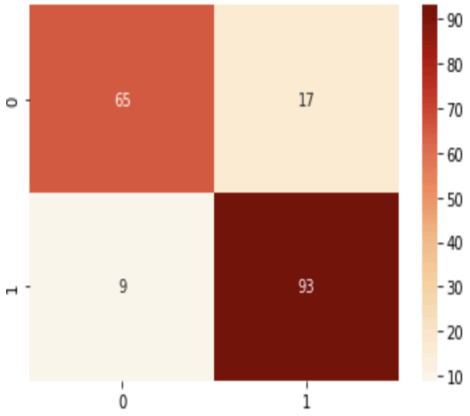




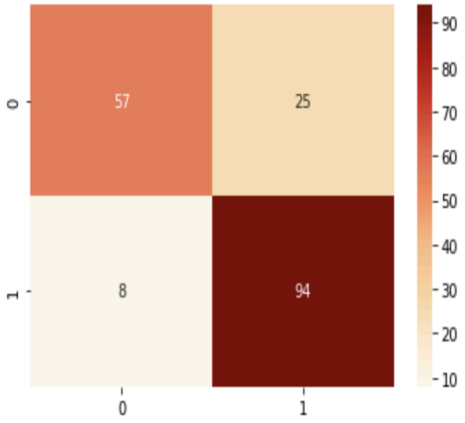
LogisticRegression



DecisionTreeClassifier



SVM



Random Forest

K-Nearest Neighbors

Gaussian Naive Bayes

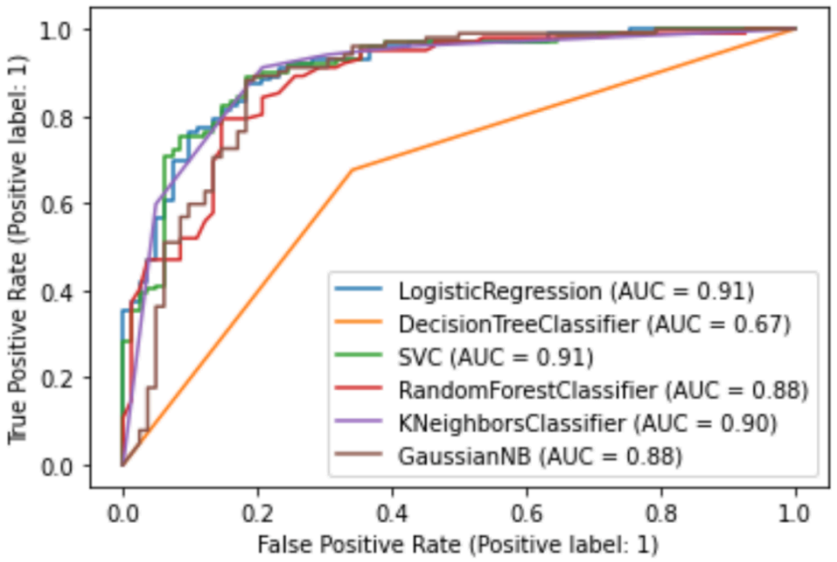
4- Results and discussion

❖ To compare all the models, we started comparing their diagnosis tests using ROC curves.

 I used this code to display all the curves in one plot.

```
disp = plot_roc_curve(LR, x_test, y_test)
plot_roc_curve(DT, x_test, y_test, ax=disp.ax_)
plot_roc_curve(clf, x_test, y_test, ax=disp.ax_)
plot_roc_curve(RF, x_test, y_test, ax=disp.ax_)
plot_roc_curve(nbrs, x_test, y_test, ax=disp.ax_)
plot_roc_curve(gnb, x_test, y_test, ax=disp.ax_)
```

Output:



As we can see from the AUC results, Logitic Regression, KNeighborsClassifier are the best algorithms in terms of prediction in our health failure dataset. And from the ROC curves, we can clearly see that the area under the curve for the LR, clf, nbrs models is larger than the area under the curve for the other three models, at all cut-offs the true positive rate is higher and the false positive rate is lower than for the 3 losing models.

- After we've compared the ROC curves for the models, we can get to their respective confusion matrices and see what we can understand from it.

From the confusion matrices, we can see which models are doing well on the database. By looking at the TP (0,0) and TN (1,1) values, the numbers must be high and balanced. (balanced: the gap between the two values is small). Also by looking at the FP (0,1), FN (1,0) values, the numbers must be small. In our example, K nearest neighbors has the best performance.

This project is implemented with the help of certain Python modules. Considering these metrices used to evaluate the models, we discovered that the K-Nearest Neighbors method provides the greatest test accuracy of 86 percent. It outperforms others since it is not constrained by the dataset's properties. The traits must be mutually independent according to Nave Bayes. The characteristics must be linearly separable for Logistic Regression to work. SVM necessitates the proper setting of parameters, and the neural network necessitates a complex and large dataset. Even while we receive an excellent result of 86% accuracy, it isn't adequate because it can't ensure that no incorrect diagnosis would be made. We hope to

demand additional dataset in order to increase accuracy because 918 dataset instances are insufficient to conduct an amazing job. In the future, we intend to use image detection to forecast illness in diverse diseases such as lung cancer.

Referencing:

Susan Post, M.D., M.S., W., 2021. Heart Problems after COVID-19. [online] Hopkinsmedicine.org. Available at: <https://www.hopkinsmedicine.org/health/conditions-and-diseases/coronavirus/heart-problems-after-covid19> [Accessed 22 November 2021].

Evander Holyfield, L., 2021. HEARTFAILURE Living with a hurting heart - ppt download. [online] Slideplayer.com. Available at: <https://slideplayer.com/slide/5774142/> [Accessed 7 December 2021].

Who.int. 2021. *Cardiovascular diseases*. [online] Available at: <https://www.who.int/health-topics/cardiovascular-diseases#tab=tab_1> [Accessed 7 December 2021].

Soni, Jyoti, et al. "Predictive data mining for medical diagnosis: An overview of heart disease prediction." International Journal of Computer Applications 17.8 (2011): 43-48.

Dangare, Chaitrali S., and Sulabha S. Apte. "Improved study of heart disease prediction system using data mining classification techniques." International Journal of Computer Applications 47.10 (2012): 44-48.

Jeremy Jordan. 2021. *Evaluating a machine learning model.*. [online] Available at: <https://www.jeremyjordan.me/evaluating-a-machine-learning-model/> [Accessed 12 December 2021].

Draelos, V., 2019. *Measuring Performance: The Confusion Matrix*. [online] Glass Box. Available at: <https://glassboxmedicine.com/2019/02/17/measuring-performance-the-confusion-matrix/> [Accessed 12 December 2021].