Predicting Heart Disease with Machine Learning
by
Ankita Guha

Background:

In the United States, Heart disease is the leading cause of death for both men and women. More than half of the deaths due to heart disease in 2015 were in men according to Centers for Disease Control and Prevention (CDC). About 655,000 people die of heart disease each year in the US., that is 1 in every 4 deaths either due to Coronary heart disease, stroke, or other cardiovascular disease. Coronary heart disease (CHD) is the most common type of heart disease, killing over 365,000 people annually in 2017. Every year about 805,000 Americans have a heart attack. Of these, 605,000 are a first heart attack and 200,000 happen in people who have already had a heart attack. Heart disease is the leading cause of death for people of most ethnicities in the United States, including African Americans, American Indian, Alaska Native, Hispanic, and white men. For women from the Pacific Islands and Asian American, American Indian, Alaska Native, and Hispanic women, heart disease is second only to cancer. Heart disease costs the United States about $219 billion each year from 2014 to 2015. This total includes the cost of healthcare services, medications, and lost productivity. [1]

Preventing heart disease is important as Heart disease is the number one cause of death worldwide. Good data-driven systems for predicting heart disease can improve the entire research and prevention process, making sure that more people can live healthy lives. To learn how to prevent heart disease it’s important first to learn reliably to detect it. The backbone of this study is a dataset from a study of heart disease that has been open to the public for many years. The study collects various measurements on patient health and cardiovascular statistics, and of course makes patient identities anonymous. Thus, developing a Machine Learning Predictive Model that could enhance the predictive power of not only historical patient health data but also with present and future patient health data, with less bias and variance in the model, is the need of the day.

Problem Statement:

Preventing heart disease is important as Heart disease is the number one cause of death worldwide. Good data-driven systems for predicting heart disease can improve the entire research and prevention process, making sure that more people can live healthy lives. To learn how to prevent heart disease we must first learn to reliably detect it. This dataset is from a study of heart disease that has been open to the public for many years. The study collects various measurements on patient health and cardiovascular statistics, and of course makes patient identities anonymous. Data is provided courtesy of the Cleveland Heart Disease Database via the UCI Machine Learning repository. Thus, developing a Machine Learning Predictive Model that could enhance the predictive power of not only historical patient health data but also with present and future patient health data, with less bias and variance in the model, is the need of the day.

Objective:

The goal was to predict the binary class heart_disease_present, which represents whether or not a patient has heart disease:

- 0 represents no heart disease present
- 1 represents heart disease present, along with the probabilities of the presence of the heart disease in the patient.

Identify statistically significant features contributing to the presence or to the absence of heart diseases in patient.

Data source Description:

[1] Heart Disease Facts, CDC

The dataset used here is from the LIVE competition, hosted by driven data.
There are 14 columns in the dataset, where the patient_id column is a unique and random identifier. The remaining 13 features are described in the section below.

- slope_of_peak_exercise_st_segment (type: int): the slope of the peak exercise ST segment, an electrocardiography read out indicating quality of blood flow to the heart
- thal (type: categorical): results of thallium stress test measuring blood flow to the heart, with possible values normal, fixed defect, reversible defect[2]
- resting_blood_pressure (type: int): resting blood pressure
- chest_pain_type (type: int): chest pain type (4 values)
- num_major_vessels (type: int): number of major vessels (0-3) colored by fluoroscopy
- fasting_blood_sugar_gt_120_mg_per_dl (type: binary): fasting blood sugar > 120 mg/dl
- resting_ekg_results (type: int): resting electrocardiographic results (values 0,1,2)
- serum_cholesterol_mg_per_dl (type: int): serum cholestoral in mg/dl
- oldpeak_eq_st_depression (type: float): oldpeak = ST depression induced by exercise relative to rest, a measure of abnormality in electrocardiograms
- sex (type: binary): 0: female, 1: male
- age (type: int): age in years
- max_heart_rate_achieved (type: int): maximum heart rate achieved (beats per minute)
- exercise_induced_angina (type: binary): exercise-induced chest pain (0: False, 1: True)

Technical Summary

Deployed various Prediction Model, using Python, and used the following libraries:

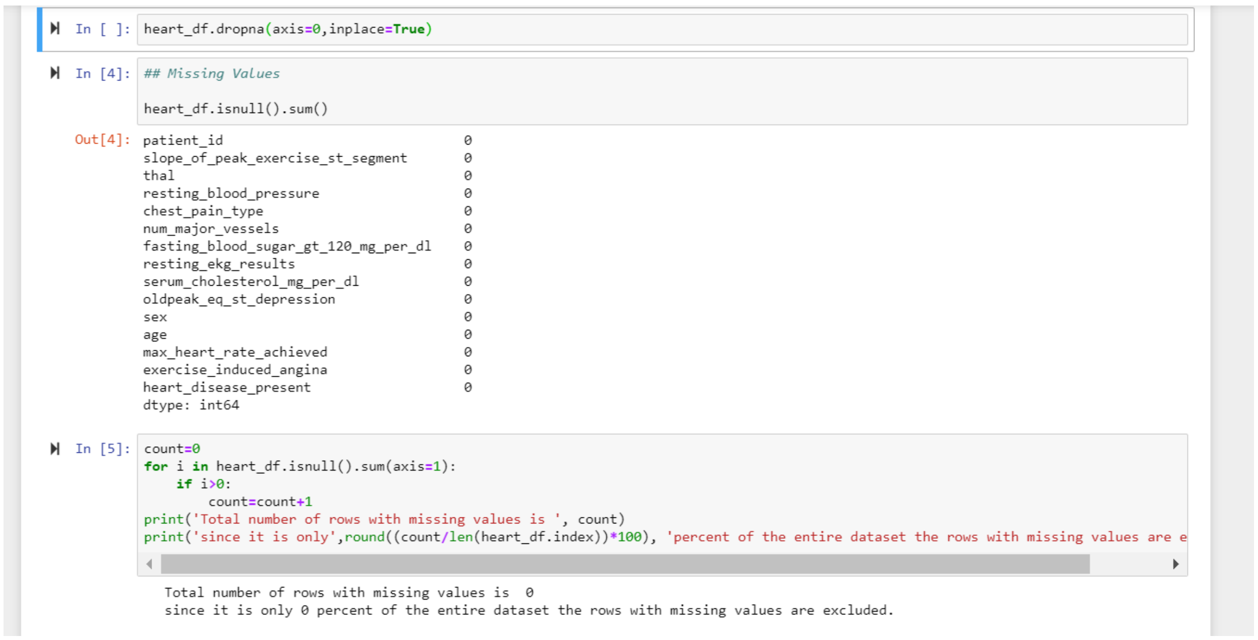
- NumPy, SciPy, Pandas for Data Wrangling and Exploratory Analysis
- Matplotlib, Seaborn for Data Visualization
- SciKit Learn, Keras, Tensorflow for building Predictive Models with Machine Learning

Also used DL4J, to see the performance of the prediction model in terms of model's performance in evaluation. The main goal here was to reduce the loss in the Prediction Model by reducing the Stochastic Gradient in the deep neural network in such a way, so that the actual predicted values are maximized and the incorrect predicted values in the Confusion Matrix are minimized. Conducted a Feature Selection based on Model outcome, as there were 15 attributes in the dataset.

Data Preparation

For Data preparation, it is interesting to notice the numerous instances of missing data (notated by a '?"). While looking through a few cases, it did not seem that simply interpolation would be able to correctly fill in the gaps. Decided to test whether or not the patient count was high enough to take out all of the rows containing empty data and still see reasonably high performance. This left us with 13 parameters for 180 patients. However, depending on the kind of Classifier I have changed some of the attributes as it can be seen from the Jupyter Notebook.

[2] If it is a "fixed" defect that would mean that it is a scar from a previous heart attack. A "reversible" defect means that there has not been a heart attack but there is a risk of one in that area.



```
In [ ]: heart_df.dropna(axis=0,inplace=True)
```

```
In [4]: ## Missing Values

        heart_df.isnull().sum()
```

```
Out[4]: patient_id                              0
        slope_of_peak_exercise_st_segment       0
        thal                                    0
        resting_blood_pressure                  0
        chest_pain_type                         0
        num_major_vessels                       0
        fasting_blood_sugar_gt_120_mg_per_dl    0
        resting_ekg_results                     0
        serum_cholesterol_mg_per_dl             0
        oldpeak_eq_st_depression                0
        sex                                     0
        age                                     0
        max_heart_rate_achieved                 0
        exercise_induced_angina                 0
        heart_disease_present                   0
        dtype: int64
```

```
In [5]: count=0
        for i in heart_df.isnull().sum(axis=1):
            if i>0:
                count=count+1
        print('Total number of rows with missing values is ', count)
        print('since it is only',round((count/len(heart_df.index))*100), 'percent of the entire dataset the rows with missing values are e
```

```
   Total number of rows with missing values is  0
   since it is only 0 percent of the entire dataset the rows with missing values are excluded.
```

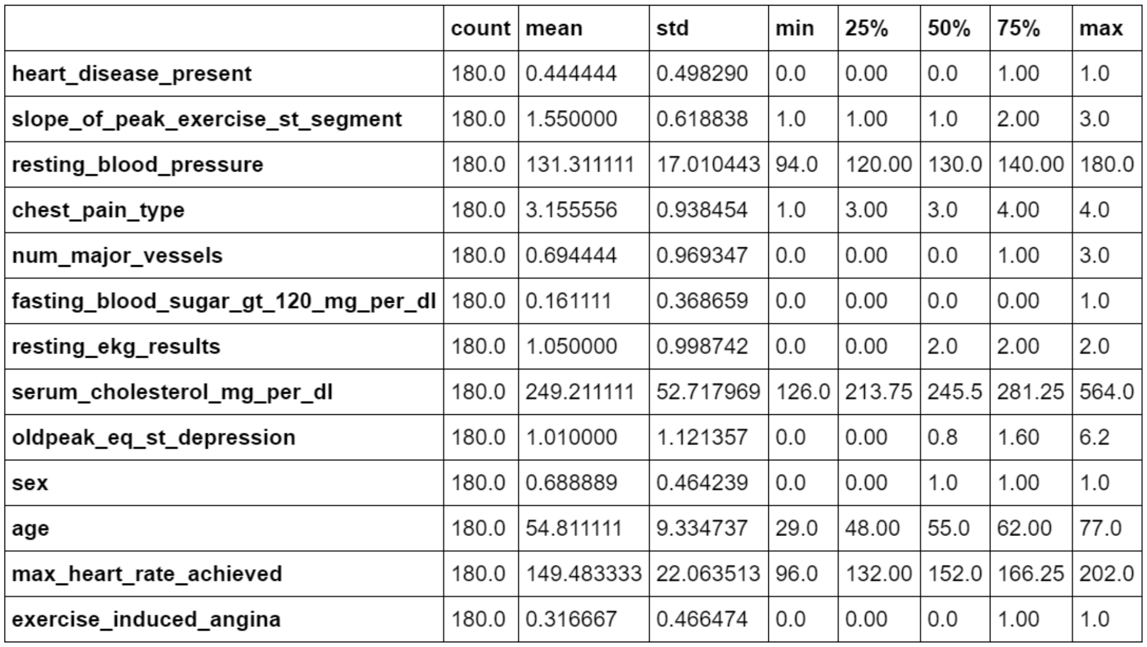
Data Preparation

```
heart_df.drop(['patient_id', 'thal'], axis=1, inplace=True)
heart_df.head()

X = heart_df.drop('heart_disease_present', axis=1)
y = heart_df['heart_disease_present']
```

After preparing the data and moving ahead with Exploratory Data analysis let’s look at statistics of all different attributes that could further help us to understand the spread/ distribution of the data and to draw conclusions from data that are subject to random variation.

:[5]:



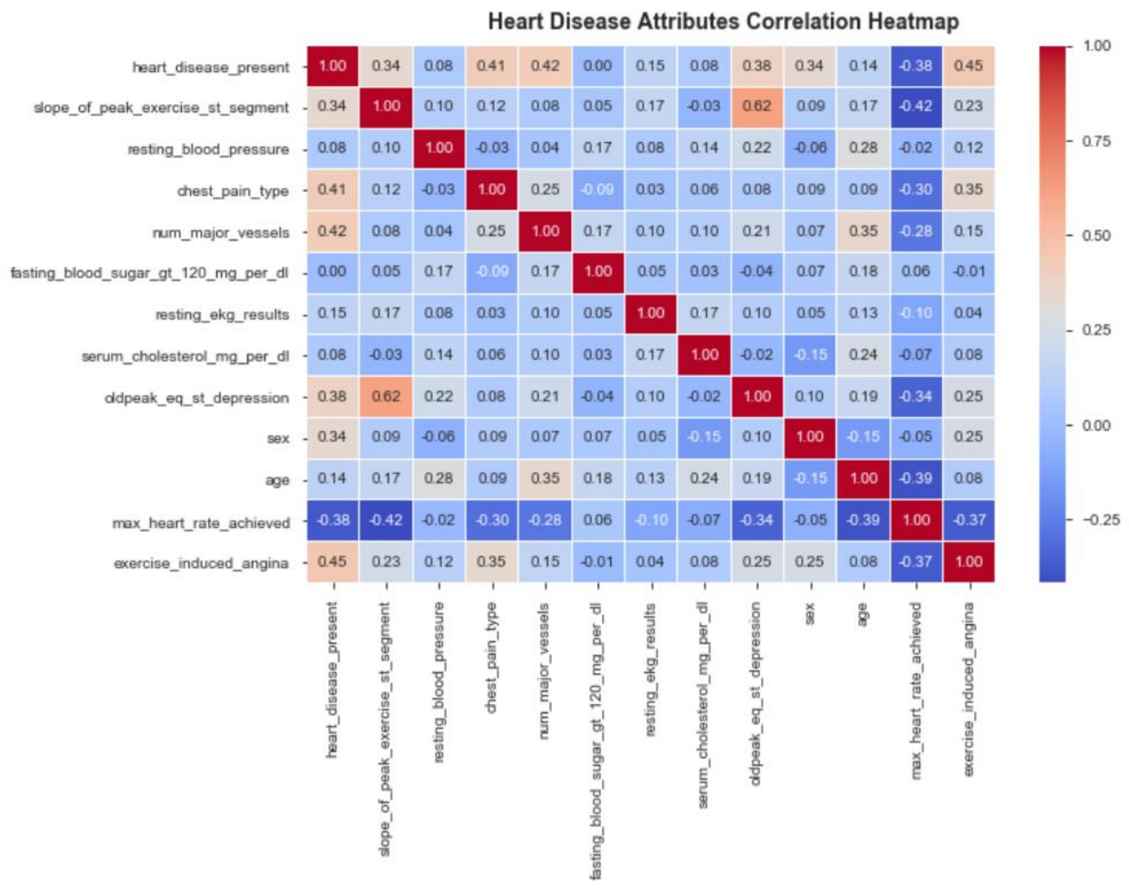
	count	mean	std	min	25%	50%	75%	max
heart_disease_present	180.0	0.444444	0.498290	0.0	0.00	0.0	1.00	1.0
slope_of_peak_exercise_st_segment	180.0	1.550000	0.618838	1.0	1.00	1.0	2.00	3.0
resting_blood_pressure	180.0	131.311111	17.010443	94.0	120.00	130.0	140.00	180.0
chest_pain_type	180.0	3.155556	0.938454	1.0	3.00	3.0	4.00	4.0
num_major_vessels	180.0	0.694444	0.969347	0.0	0.00	0.0	1.00	3.0
fasting_blood_sugar_gt_120_mg_per_dl	180.0	0.161111	0.368659	0.0	0.00	0.0	0.00	1.0
resting_ekg_results	180.0	1.050000	0.998742	0.0	0.00	2.0	2.00	2.0
serum_cholesterol_mg_per_dl	180.0	249.211111	52.717969	126.0	213.75	245.5	281.25	564.0
oldpeak_eq_st_depression	180.0	1.010000	1.121357	0.0	0.00	0.8	1.60	6.2
sex	180.0	0.688889	0.464239	0.0	0.00	1.0	1.00	1.0
age	180.0	54.811111	9.334737	29.0	48.00	55.0	62.00	77.0
max_heart_rate_achieved	180.0	149.483333	22.063513	96.0	132.00	152.0	166.25	202.0
exercise_induced_angina	180.0	0.316667	0.466474	0.0	0.00	0.0	1.00	1.0

Exploratory Data analysis:

Tried to understand the relationship between different attributes as to whether or not they have any correlation or not.

The relationships between different variables can be visualized using the namespace plot from matplotlib. Scatter plot used herein helps visualizing relationships between two numerical variables, and the box plot used for visualizing relationships between one numerical variable and one categorical variable. The complex conditional plot was used to visualize many variables in a single visualization.

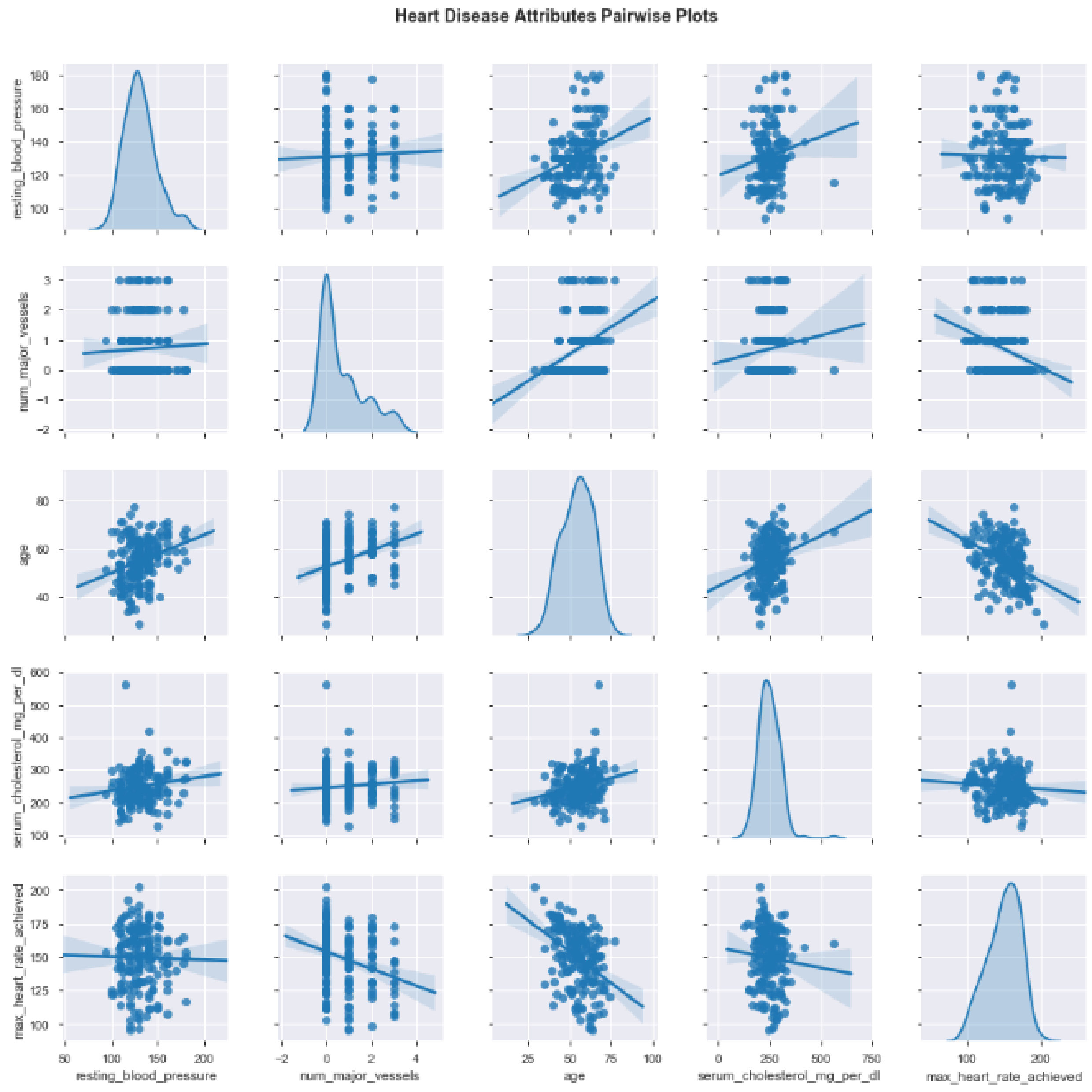
From the Correlation Heatmap as shown, it gave an idea of the relationship amongst the various continuous and categorical variables.



Pairplot:

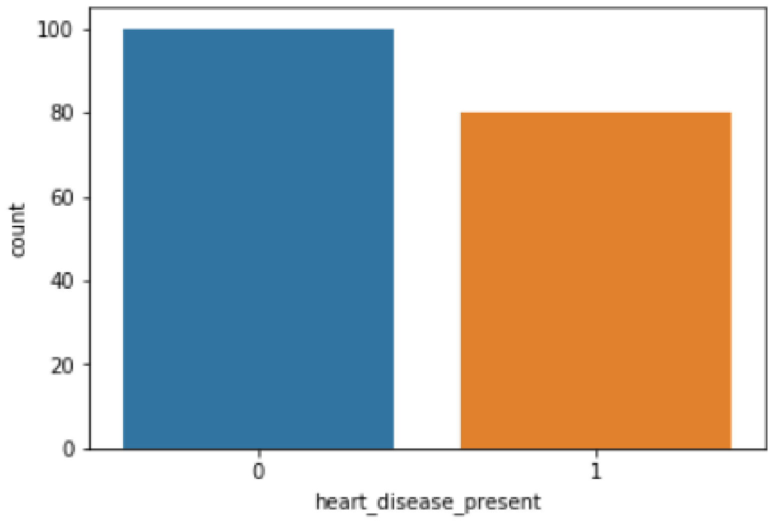
Below mentioned pairplot gave us an overview of the various relationships for each pairs of columns within our dataframe. As it can be seen the Histograms Distributions are from the numeric variables. The Scatterplots gives an idea of the various Categorical Variables as well as the relationship between two Numeric Variables.

From the pairplot of the whole data we got an idea of the various variables and their respective features. Then plotted some of the numerical variables in another pairplot and tried to get an idea of the various relationships amongst them.

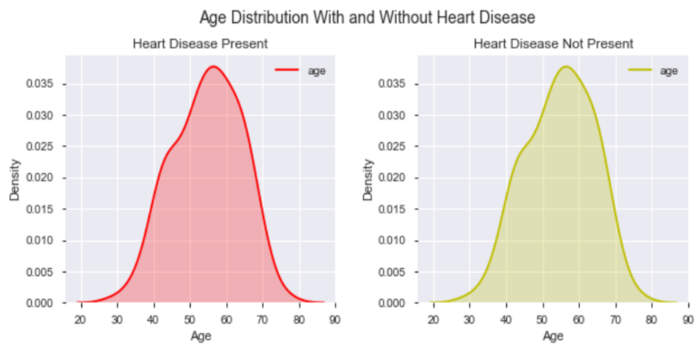


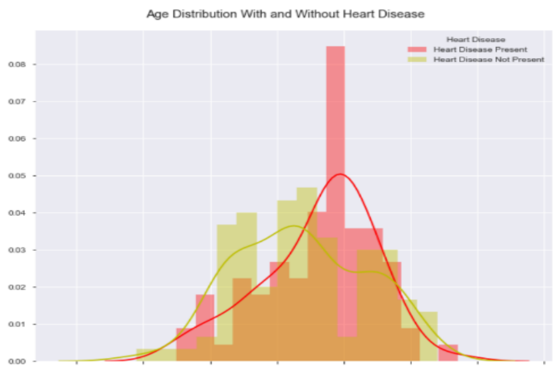
Frequency Plot:

From the frequency plot of heart disease below, we see that the two classes (‘Heart Disease’ and ‘No Heart Disease’) are approximately balanced, with 45% of observations having heart disease and the remaining population not having heart disease.



While trying to understand the distribution of data we tried different types of plotting before which we actually converted variables from binary to categorical type for our ease and then plotted the data using different types to understand the distribution of data graphically. For this purpose, I used stacked bar plot, Density plot and Histogram. Histograms are the ones that gave us the best results and we now see that our data is somewhat normally distributed as shown below:





Data Modelling:

A. Logistic Regression:

Logistic regression is a type of regression analysis in statistics used for prediction of outcome of a categorical dependent variable from a set of predictor or independent variables. In logistic regression the dependent variable is always binary. Logistic regression is mainly used for prediction and also calculating the probability of success.

For Logistic Regression Model we trained and tested our data to fit the model and see if we can get similar prediction on the new test data set. Here we see that our F1 Score is 78%
Next, we ran the confusion matrix to check how correct our predictions are:
It was evident from our Confusion Matrix that we have 24, and 23 correct predictions and 3, 10 in correct predictions.

ROC Curve:

The ROC curve is thus the sensitivity as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability on the x-axis. Next let’s see exploring different attributes like Exercise induced Angina, chest pain type, Fasting Blood Sugar. Feature Selection: Performed Feature Elimination to get the predictions with the best possible attributes:

Recursive feature elimination

Given an external estimator that assigns weights to features, recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a coef_attribute or through a feature importance attribute. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

Feature ranking with Recursive Feature Elimination and Cross-Validation

We used RFECV to perform RFE in a cross-validation loop to find the optimal number or the best number of features from out of all the 14 features. Hereafter a recursive feature elimination applied on logistic regression with automatic tuning of the number of features selected with cross-validation.

Now, we had used 11 features, with which we built our Model. However, from Feature Ranking we can see that the maximum Model performance happens when the Model is fit with 10 Features selected. And it seems like serum_cholesterol_mg_per_dl does not significantly contribute to the Model's performance and hence it is dropped from the Model. But on checking the model accuracy we saw that the model accuracy decreased and then we tried to Fold Cross Validation with which we see increase in Model accuracy to 83%.

To check the effect of adding back the attribute that we removed earlier serum_cholesterol_mg_per_dl, it can be seen that it doesn’t make any difference.

Grid Search Using Multi scorers Simultaneously

Applied many tasks together for more in-depth evaluation like grid search using cross-validation based on k-folds repeated many times, that can be scaled or not with respect to many scorers and tuning on parameter for a given estimator!

B. SVM, Decision Tree and Random Forest

After running Logistic Regression for the purpose of better understanding the Model’s performance, decided to run some of the other Model in order to see the Model’s performance. After preparing the data in terms of feeding into the Model, dropped some of the columns, to see the Model’s performance with simple SVM (Support Vector Machine).

The Model’s performance of Linear SVM seemed to have increased significantly from Logistic Regression. Usually, hyper tuning the parameters helps to improve the Model’s performance, but in this case, it seems like the Model’s performance did not improve.

Next tried to run Polynomial Kernel SVM, however the Model’s performance seemed to have decreased to 67% now.
With Gaussian Kernel SVM, the Model’s performance has seemed to have increased to 83%. With Sigmoid Kernel the performance has seemed to have the same performance to 83%. With Decision Tree also the Model’s performance seems to have increased to 83%. However, with Random Forest the Model’s performance seemed to have increased to 84%.

As of now, it seems like Random Forest has performed best in terms of Model’s performance.

C. Naïve Bayes Classifier

After cleaning and preparing the data for the purpose of feeding the data into Naïve Bayes Classifier, had to convert the variable "Thal" into Categorical Variable. After feeding the data variable models, into the Gaussian Classifier it looks like that this Model performed the best out of all the Model, with a whopping performance of 91%. The probability of predicting the heart disease with this Classifier is 41.67% and the probability of predicting the absent of heart disease with this Classifier is 58.33%.

Next step tried to see if certain features or attributes contributed towards the presence or absence of heart disease or not. Seems like, the **Maximum Heart Rate achieved during the stress test** has an impact on the presence and absence of heart disease in patient. Model's performance seems to have reached to 58.33%. The impact of **Resting Blood Pressure** also seemed to have an impact on the presence and the absence of heart disease. The Model's performance seemed to have decreased to 41.67% this time when taking resting blood pressure's contribution into the account of predicting the presence or the absence of heart disease. Next, in the final step tried to see the impact of the **Blood Cholesterol** in varying the performance of the Model in predicting the presence or the absence of heart disease in patient. The Model's performance seemed to have improved a little to 44.44%.

D. Multilayer Perceptron Neural Network

The next Model, that we tried running is the Multilayer Perceptron Neural Network, with 13 hidden layers of neural network and ran the iterations for 500 times. On predicting the Model's performance, it seems like the Model classifies the data correctly the presence or the absence of the heart disease with the specific demarcation line indicating that the Model's performance to 80%, indicating that in 80% of the cases Model classifies the presence and the absence of heart diseases in patient. And on our prediction set, it seems like the Model, performs 81% accurately in predicting the presence of heart disease in patient.

E. DL4J, Multilayer Perceptron Neural Network

I) Source Code:

```java
package org.deeplearning4j.examples.dataexamples;

import org.datavec.api.records.reader.RecordReader;
import org.datavec.api.records.reader.impl.csv.CSVRecordReader;
import org.datavec.api.split.FileSplit;
import org.datavec.api.util.ClassPathResource;
import org.deeplearning4j.datasets.datavec.RecordReaderDataSetIterator;
import org.deeplearning4j.eval.Evaluation;
import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.layers.DenseLayer;
import org.deeplearning4j.nn.conf.layers.OutputLayer;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.nn.weights.WeightInit;
import org.deeplearning4j.optimize.listeners.ScoreIterationListener;
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.DataSet;
import org.nd4j.linalg.dataset.SplitTestAndTrain;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.dataset.api.preprocessor.DataNormalization;
import org.nd4j.linalg.dataset.api.preprocessor.NormalizerStandardize;
import org.nd4j.linalg.learning.config.Sgd;
import org.nd4j.linalg.lossfunctions.LossFunctions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HeartData {
    private static Logger log = LoggerFactory.getLogger(HeartData.class);

    public static void main(String[] args) throws  Exception {

        //First: get the dataset using the record reader. CSVRecordReader handles loading/parsing
        int numLinesToSkip = 0;
        char delimiter = ',';
        RecordReader recordReader = new CSVRecordReader(numLinesToSkip,delimiter); //recordReader is the Variable. Instance of the Class.
CSVRecordReader is the Class that takes 2 arguments that are inside the parenthesis
        recordReader.initialize(new FileSplit(new ClassPathResource("HeartData.txt").getFile())); //Method is the initialize. Filesplit is the
Method to split the data to Training & Test Data

        //Second: the RecordReaderDataSetIterator handles conversion to DataSet objects, ready for use in neural network
        int labelIndex = 13;     //Since the labels for the Heart starts at the last column, out of the Total 14 Columns in the CSV File.
        int numClasses = 2;     //Classes have integer values 0, 1. Total 2 classes (starting from 0 to 1) in the heart data set.
        int batchSize = 180;    //The number of training instances used in 1 iteration. Heart data set: 180 examples total. We are loading all
of them into one DataSet (not recommended for large data sets)

        DataSetIterator iterator = new RecordReaderDataSetIterator(recordReader,batchSize,labelIndex,numClasses);
        DataSet allData = iterator.next(); //it would point to the next record.
        allData.shuffle(); // Shuffle the values in a single column, make predictions using the resulting data-set. Use these predictions and
the true target values to calculate how much the loss function suffered from shuffling. That performance deterioration measures the importance
of the variable you just shuffled.
        SplitTestAndTrain testAndTrain = allData.splitTestAndTrain(0.80);  //Use 80% of data for training

        DataSet trainingData = testAndTrain.getTrain();
        DataSet testData = testAndTrain.getTest();

        //We need to normalize our data. We'll use NormalizeStandardize (which gives us mean 0, unit variance, which is a Standard
Normalization):
        DataNormalization normalizer = new NormalizerStandardize();
        normalizer.fit(trainingData);           //Collect the statistics (mean/stdev) from the training data. This does not modify the input
data
        normalizer.transform(trainingData);     //Apply normalization to the training data
        normalizer.transform(testData);         //Apply normalization to the test data. This is using statistics calculated from the
*training* set


        final int numInputs = 13; //columns
        int outputNum = 2; // 2 levels
        long seed = 6;


        log.info("Build model....");
        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
            .seed(seed)
            .activation(Activation.TANH)
            .weightInit(WeightInit.XAVIER)
            .updater(new Sgd(0.1))
            .l2(1e-4)
            .list() // 3 layers used here although we are just predicting 2 classes here, Heart Disease Present or Not
            .layer(0, new DenseLayer.Builder().nIn(numInputs).nOut(2) //Input Layer
                .build())
            .layer(1, new DenseLayer.Builder().nIn(2).nOut(2)
                .build())
            .layer(2, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                .activation(Activation.SOFTMAX)
                .nIn(2).nOut(outputNum).build())
            .backprop(true).pretrain(false) //pre-train is used to specify a certain value of weights so that weights get converged quickly
            .build();

        //run the model
        MultiLayerNetwork model = new MultiLayerNetwork(conf);
        model.init();
        model.setListeners(new ScoreIterationListener(100)); // ScoreIterationListener will simply print the current error score for your
network.Iterations will run for < 4000 times & starting from 0, each Iteration will jump to another 100 times.

        for(int i=0; i<4000; i++ ) {
            model.fit(trainingData);
        }

        //evaluate the model on the test set
        Evaluation eval = new Evaluation(2); //Creates an Evaluation object with 2 Classes
        INDArray output = model.output(testData.getFeatures());
        eval.eval(testData.getLabels(), output);
        log.info(eval.stats());
        System.out.println(eval.accuracy());
        System.out.println(eval.precision());
        System.out.println(eval.recall());

        //eval.getConfusionMatrix(); Added this one to check
        //eval.getConfusionMatrix().toHTML();

    }
}
```

II) **Output:**

```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Community Edition 2018.2.3\lib\idea_rt.jar=50419:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
2018.2.3\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
2018.2.3\lib\idea_rt.jar" com.intellij.rt.execution.CommandLineWrapper
C:\Users\ANKITA~1\AppData\Local\Temp\idea_classpath1625796545
org.deeplearning4j.examples.dataexamples.HeartData
o.n.l.f.Nd4jBackend - Loaded [CpuBackend] backend
o.n.n.NativeOpsHolder - Number of threads used for NativeOps: 2
o.n.n.Nd4jBlas - Number of threads used for BLAS: 2
o.n.l.a.o.e.DefaultOpExecutioner - Backend used: [CPU]; OS: [Windows 10]
o.n.l.a.o.e.DefaultOpExecutioner - Cores: [4]; Memory: [1.7GB];
o.n.l.a.o.e.DefaultOpExecutioner - Blas vendor: [MKL]
o.d.e.d.HeartData - Build model....
o.d.n.m.MultiLayerNetwork - Starting MultiLayerNetwork with WorkspaceModes set to [training: ENABLED;
inference: ENABLED], cacheMode set to [NONE]
o.d.o.l.ScoreIterationListener - Score at iteration 0 is 0.6698911426105579
o.d.o.l.ScoreIterationListener - Score at iteration 100 is 0.3546269266906897
o.d.o.l.ScoreIterationListener - Score at iteration 200 is 0.28959313913574974
o.d.o.l.ScoreIterationListener - Score at iteration 300 is 0.27730768514446646
o.d.o.l.ScoreIterationListener - Score at iteration 400 is 0.2695445798750703
o.d.o.l.ScoreIterationListener - Score at iteration 500 is 0.26356719199827405
o.d.o.l.ScoreIterationListener - Score at iteration 600 is 0.25846299885380264
```

```
o.d.o.l.ScoreIterationListener - Score at iteration 700 is 0.2538940044763914
o.d.o.l.ScoreIterationListener - Score at iteration 800 is 0.24973638903301007
o.d.o.l.ScoreIterationListener - Score at iteration 900 is 0.24592095589941318
o.d.o.l.ScoreIterationListener - Score at iteration 1000 is 0.24237882665873522
o.d.o.l.ScoreIterationListener - Score at iteration 1100 is 0.23902718958176652
o.d.o.l.ScoreIterationListener - Score at iteration 1200 is 0.23576760435524025
o.d.o.l.ScoreIterationListener - Score at iteration 1300 is 0.23255519653640352
o.d.o.l.ScoreIterationListener - Score at iteration 1400 is 0.22947993088790236
o.d.o.l.ScoreIterationListener - Score at iteration 1500 is 0.22663139999478654
o.d.o.l.ScoreIterationListener - Score at iteration 1600 is 0.2240000137755246
o.d.o.l.ScoreIterationListener - Score at iteration 1700 is 0.22156871283970367
o.d.o.l.ScoreIterationListener - Score at iteration 1800 is 0.21942589850042157
o.d.o.l.ScoreIterationListener - Score at iteration 1900 is 0.21765542633384077
o.d.o.l.ScoreIterationListener - Score at iteration 2000 is 0.21620884117698086
o.d.o.l.ScoreIterationListener - Score at iteration 2100 is 0.21499306475277086
o.d.o.l.ScoreIterationListener - Score at iteration 2200 is 0.21394062357709576
o.d.o.l.ScoreIterationListener - Score at iteration 2300 is 0.2130105490892576
o.d.o.l.ScoreIterationListener - Score at iteration 2400 is 0.21217718483971584
o.d.o.l.ScoreIterationListener - Score at iteration 2500 is 0.2114229277876867
o.d.o.l.ScoreIterationListener - Score at iteration 2600 is 0.21073493012203795
o.d.o.l.ScoreIterationListener - Score at iteration 2700 is 0.21010341701024926
o.d.o.l.ScoreIterationListener - Score at iteration 2800 is 0.20952046800799726
o.d.o.l.ScoreIterationListener - Score at iteration 2900 is 0.20897980510654035
o.d.o.l.ScoreIterationListener - Score at iteration 3000 is 0.20847617018980408
o.d.o.l.ScoreIterationListener - Score at iteration 3100 is 0.20800516608290226
o.d.o.l.ScoreIterationListener - Score at iteration 3200 is 0.2075630578617574
o.d.o.l.ScoreIterationListener - Score at iteration 3300 is 0.20714661391532474
o.d.o.l.ScoreIterationListener - Score at iteration 3400 is 0.20675298674470288
o.d.o.l.ScoreIterationListener - Score at iteration 3500 is 0.20637977916441036
o.d.o.l.ScoreIterationListener - Score at iteration 3600 is 0.20602481915524515
o.d.o.l.ScoreIterationListener - Score at iteration 3700 is 0.2056862260832099
o.d.o.l.ScoreIterationListener - Score at iteration 3800 is 0.20536221201791946
o.d.o.l.ScoreIterationListener - Score at iteration 3900 is 0.20505122744316562
o.d.e.d.HeartData -

========================Evaluation Metrics========================
 # of classes:    2
 Accuracy:        0.8333
 Precision:       0.8545
 Recall:          0.8095
 F1 Score:        0.7692
Precision, recall & F1: reported for positive class (class 1 - "1") only


=========================Confusion Matrix=========================
  0  1
-------
 20  1 | 0 = 0
  5 10 | 1 = 1

Confusion matrix format: Actual (rowClass) predicted as (columnClass) N times
==================================================================
0.8333333333333334
0.9090909090909091
0.6666666666666666

Process finished with exit code 0
```

Explanation:

The Heart Data before feeding into DL4J for building the Multilayer Perceptron Neural Network is cleaned after running and building different Models in Python. All other Categorical data variables such as the "Thal" (which is a stress test measuring blood flow to the heart and is classified into: normal, fixed_defect, reversible_defect) is converted into Numeric Categorical Variables (0, 1, 2) for the process of building the Model data into DL4J. Precision and Recall both are at a range higher than 80% indicating that the Model's overall performance on predicting is pretty good. Recall/Sensitivity measures how well the Model avoids False Negative and Precision/Specificity gives an idea on how well the Model performs when the same measurements are repeated, an indication on how the Model responses to False Positive. A Model with both higher Recall and Precision gives an indication of good Model performance.

We have 2 data classifier that are used for predicting the Presence or absence of heart disease in patient.
0: Patient does not have a heart disease
1: Patient have heart disease

Confusion Matrix:

1. True Positive: The Classifier has predicted 20 Classes correctly in predicting the fact that the Patient does not have a heart disease (0).
2. True Negative: The Classifier has predicted 10 Classes correctly in predicting the fact that the Patient does not have a heart disease (0).
3. False Positive: The Classifier however has predicted 5 Classes incorrectly as 0, (indicating Patient does not have a heart disease), whereas actually the Patient has a heart disease (1).
4. False Negative: The Classifier has predicted just 1 Class incorrectly as 1, (indicating Patient have a heart disease), whereas actually the Patient did not have a heart disease (0).

Key Takeaway:

- Higher the number of iterations, higher is the accuracy.
- The accuracy as well as the Confusion Matrix Classification Report changes on changing the train and test data split.
- Even changing the set of iterations has changed some of the Model's accuracy, Precision and Recall.
- As compared to Python, there are few pros and cons that I have experienced. They are as follows:

Pros of DL4J Over Python:

- The Model performances can be seen directly in the Terminal window and it helps in understanding the various layers of Neural Network that has to be tuned in order to get the optimal findings.

Cons of DL4J Over Python:

- Could not find a possible way of visualizing the data frame for better understanding.
- Could not perform Exploratory Data Analysis.
- Could not preprocess the data before feeding into the Neural Network.

<u>Table with all the Model's Performance</u>

Serial Number	Model Name	Performance (Accuracy)
1.	Simple Logistic Regression	78%
2.	Feature Selection Logistic Regression	73%
3.	10Fold Cross Validation Logistic Regression	81%
4.	Grid Search CV, Logistic Regression	83%
5.	Simple SVM	89%
6.	Polynomial Kernel SVM	67%
7.	Gaussian Kernel	50%
8.	Sigmoid Kernel	50%
9.	Decision Tree	73%
10.	Random Forest	84%
11.	Naïve Bayes Classifier	91.67%
12.	Multilayer Perceptron Neural Network	81%
13.	DL4J, Multilayer Perceptron Neural Network	83%

Challenges:

Since the dataset was small it resulted in overfitting the model and we also faced a lot of challenges in terms of tuning the hyperparameters and reducing the Stochastic Gradient. Outliers skewed the data and ignoring them could have caused Overfitting or Underfitting of the Model parameters. Hyperparameters were tuned only in case of SVM. There are a lot of way by which each Model's Hyperparameters could have been tuned and optimized, but due to less RAM power and processor speed could not deploy all of the optimization techniques.

Conclusion:

In this project, we carried out an experiment to find the predictive performance of heart disease with different classifiers and Model. Selected popular classifiers considering their qualitative performance for the experiment. Naive Bayes Classifier gave the best performance as compared to all the other Models. In order to compare the classification performance of machine learning algorithms, classifiers are applied on same data and results are compared on the basis of misclassification and correct classification rate. According to experimental results in table, it can be concluded that Naïve Bayes classifier is the best as compared to Support Vector Machine, Logistic and Random Forest for our Heart Disease prediction. However, the prediction results might differ based on the different datasets, different random seed value, as well as on considering different loss functions, activation functions and other hyperparameters tuning.
The final file that predicts on the test data is under the name of "Evaluation.csv" that contains the predicted values of heart diseases on the test data.
However, to compare the comparative performance of all the classifiers, it would be better to perform more experimentation with the Hyperparameters tuning, Optimization on several other datasets to draw general conclusion on the performance of each Model. For this study, Naïve Bayes Classifier seems to perform the best, which might not be the case when tested with other datasets.