Group Number: 326

Cardio Disease Prediction

First Name	Last Name	Email (hawk.iit.edu)	Student ID
Vaishnavi	Chaudhari	vchaudhari@hawk.iit.edu	A20446094
Megha	Shrivastava	mshrivastava1@hawk.iit.edu	A20450886

Table of Contents

1. Introduction

In today's world, there is a substantial increase in heart diseases. The data set that we have selected predicts whether an individual will suffer from cardio disease or not. The prediction of the disease is impacted by various features. The data set used here has information related to the cardio disease. According to the dataset, the presence of the cardio disease is predicted using id number, age, gender, height, weight, systolic blood pressure, diastolic blood pressure, cholesterol level, glucose, smoking habits, alcohol consumption, and day to day activities. The model will be predicting, classifying, and clustering the data according to the problem statement. The model presented can be useful to predict if an individual following certain trends and habit is prone to cardio disease or not.

2. Data Sets

The dataset is a cardio disease dataset. The data can be referred from:
https://www.kaggle.com/raminhashimzade/cardio-disease.
The data frame with 70000 rows and 14 variables- 13 feature variables and 1 predict variable. The variables present in the data frame are: **id** (number of observations) , **age** (age of the patient in days), **age** (age of the patient in years), **gender** (gender of the patient where 1 – women, 2 - men), **height** (height of the patient in cm), **weight** (weight of the patient in kg), **ap_hi** (systolic blood pressure of the patient), **ap_lo** (diastolic blood pressure of the patient), **cholesterol** (cholesterol level of the patient where 1:normal, 2:above normal, 3:well above normal), **gluc** (glucose level of the patient where 1:normal, 2:above normal, 3:well above normal), **smoke** (whether patient smokes or not where 0 = no, 1= yes), **alco** (amount of alcohol consumption of the patient which is a binary feature where 0 = no, 1 = yes), **active** (Active live represented by binary feature where 0 = if the patient have passive life, 1 = if the patient have active life), **cardio** (It is the target variable represented by 0 and 1, 0 = if the patient does not have cardio disease and 1 = if the patient have cardio disease).

	id	age_days	age_year	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	50.391781	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	55.419178	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	51.663014	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	48.282192	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	47.873973	1	156	56.0	100	60	1	1	0	0	0	0

The data type of the variables
Id, age, height, weight, ap_hi, ap_lo: Discrete variables (Quantitative datatype)
Gender, cholesterol, gluc: Ordinal variable (Qualitative datatype)
Smoke, alco, active, cardio: Binary variable (Qualitative datatype)

3. Research Problems

The problem statement is: **To predict whether the patient will have a cardio disease or not based on the attribute or the features associated with it. Classifying the patient's data into patients with the cardio disease and without cardio disease and grouping them together. Trying out different classification algorithms to find the model with the best classifying abilities.** If it is predicted that the patient will suffer from the cardio disease, then the value is 1 and if it is predicted that the patient will not suffer from the cardio disease then the value is 0. In the given dataset we have several factors that affect the cause of cardio disease, so it is important to find out how much each attribute has a contribution to the changing value of the predicted variable.

4. Potential Solutions

To address the problem mentioned in the problem statement we need different **classification models** that can be used to classify the values of the cardio variable with changing values of the x variables (attribute values). To **classify** the patients, we need to build a classification of the data set selected. For building these models we need to preprocess the data variables according to each model. To get the best model we need to select those features which have the maximum effect on the cardio variable. The models will be evaluated on the basis of the values of **accuracy**, **precision** and **recall**.

For this dataset, we have one **dependent** variable which will be predicting the values of the possibility of the patient having the cardio disease. The variable we are going to predict will be the **cardio** variable.
The **independent variables** for this predictive model will be the attributes or the features used by one to determine the condition of the patient. The independent variables present in the dataset are – Id, age, height, weight, ap_hi, ap_lo, Gender, cholesterol, gluc, Smoke, alco, active.

5. Evaluations

5.1 Methods and Process

Descriptive analysis:
The initial stage of building a model is getting to know the data. Descriptive analysis has been performed on the data and the outcomes from that are:

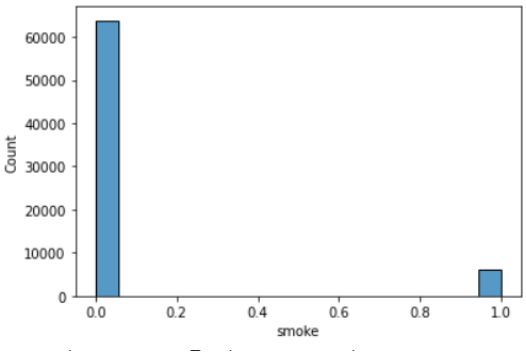
	age_year	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
count	69840.000000	69840.000000	69840.000000	69840.000000	69840.000000	69840.000000	69840.000000	69840.000000	69840.000000	69840.000000	69840.000000	69840.000000
mean	53.342057	1.349470	164.410782	74.153489	126.712829	93.788846	1.366967	1.226589	0.088216	0.053723	0.803637	0.499570
std	6.758310	0.476806	7.918345	14.205343	18.053400	108.290828	0.680363	0.572388	0.283611	0.225471	0.397249	0.500003
min	29.583562	1.000000	120.000000	10.000000	1.000000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	48.400000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
50%	53.983562	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
75%	58.430137	2.000000	170.000000	82.000000	140.000000	90.000000	2.000000	1.000000	0.000000	0.000000	1.000000	1.000000
max	64.967123	2.000000	200.000000	160.000000	401.000000	1900.000000	3.000000	3.000000	1.000000	1.000000	1.000000	1.000000

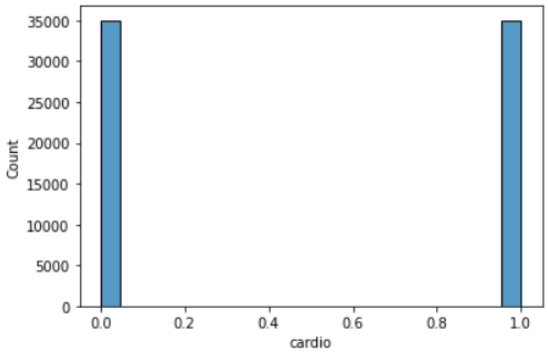
The standard deviation, mean and the quantile values of the feature variables are mentioned in the descriptive analysis. The data is of int64 type and some of the variables are of float datatype. There are 70000 rows. The data does not have any null values.

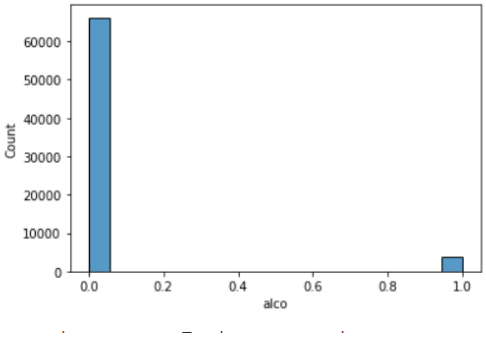
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   age_days     70000 non-null  int64
 1   age_year     70000 non-null  float64
 2   gender       70000 non-null  int64
 3   height       70000 non-null  int64
 4   weight       70000 non-null  float64
 5   ap_hi        70000 non-null  int64
 6   ap_lo        70000 non-null  int64
 7   cholesterol  70000 non-null  int64
 8   gluc         70000 non-null  int64
 9   smoke        70000 non-null  int64
 10  alco         70000 non-null  int64
 11  active       70000 non-null  int64
 12  cardio       70000 non-null  int64
dtypes: float64(2), int64(11)
memory usage: 6.9 MB
```

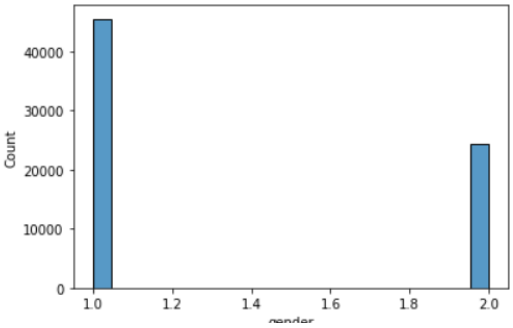
The variables can be visualized as follows:

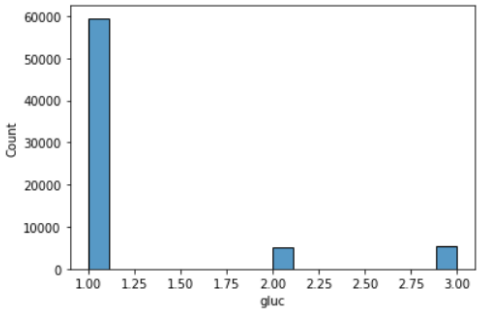
1. **Smoke:** It is a binary variable, having biased data. These are many datapoints having 0 (non-smokers) as their values than 1 (smokes).
2. **Cardio:** This is our **label variable** which is being classified. It is a variable with binary values as 1 (cardiac disease present) and 0 (not having a cardiac diseases). The data has exactly equal number of observations with 1 and 0 values. Which tells us that the **dataset** is good and is **not-biased**.
3. **Alco:** It is a binary variable, having biased data. There are many datapoints having 0 (non-alcoholic) as their values than 1 (alcoholic).
4. **Gender:** It is a binary variable, having almost un-biased data. There are same datapoints having 2 (male) as their values and datapoint having 1 (female).
5. **Gluc:** It is a variable having 3 values as 1, 2 and 3. the levels are denoted by 1, 2 and 3. The values for the variables are distributed in an unequal form. The data has more observations with values having 1 glucose level.
6. **Active:** It is a binary variable, having biased data. There are many datapoints having 1 (active) as their values than 0 (in-active).
7. **Cholesterol:** It is a variable having 3 values as 1, 2 and 3. the levels are denoted by 1, 2 and 3. The values for the variables are distributed in an unequal form. The data has more observations with values having 1 cholesterol level.
8. **Age_year:** The values are properly distributed and there are almost no outliers for this variable.
9. **Height:** The values are properly distributed but there are some outliers for this variable. The outliers could be because of some error while noting or may be some medical exceptions.
10. **Weight:** The values are properly distributed but there are some outliers for this variable. The outliers could be because of some error while noting or may be some medical exceptions.
11. **Ap_hi:** The values are properly distributed but there are some outliers for this variable. These variables are lying very far from the other data recorded. The outliers could be because of some error while noting or may be some medical exceptions.
12. **Ap_lo:** The values are properly distributed but there are some outliers for this variable. These variables are lying very far from the other data recorded. The outliers could be because of some error while noting or may be some medical exceptions.

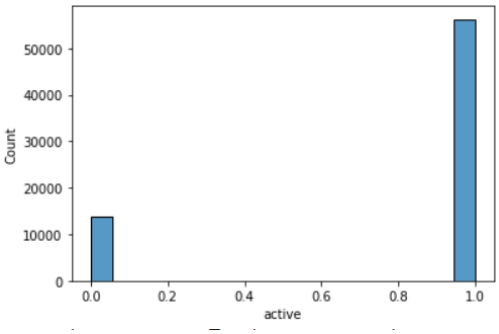

Count
smoke

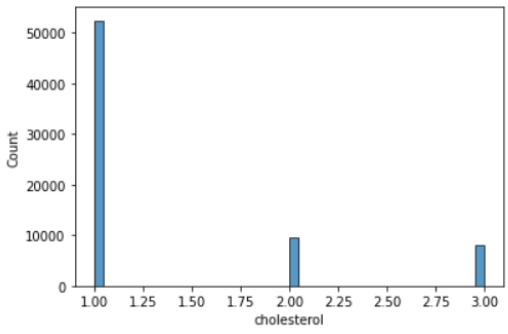

Count
cardio

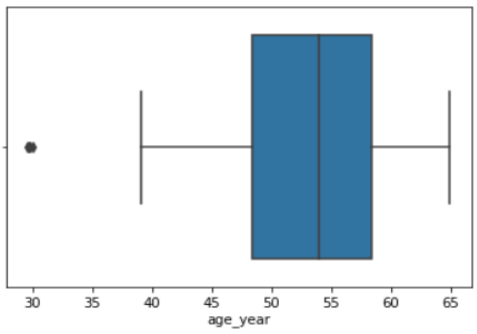

Count
alco

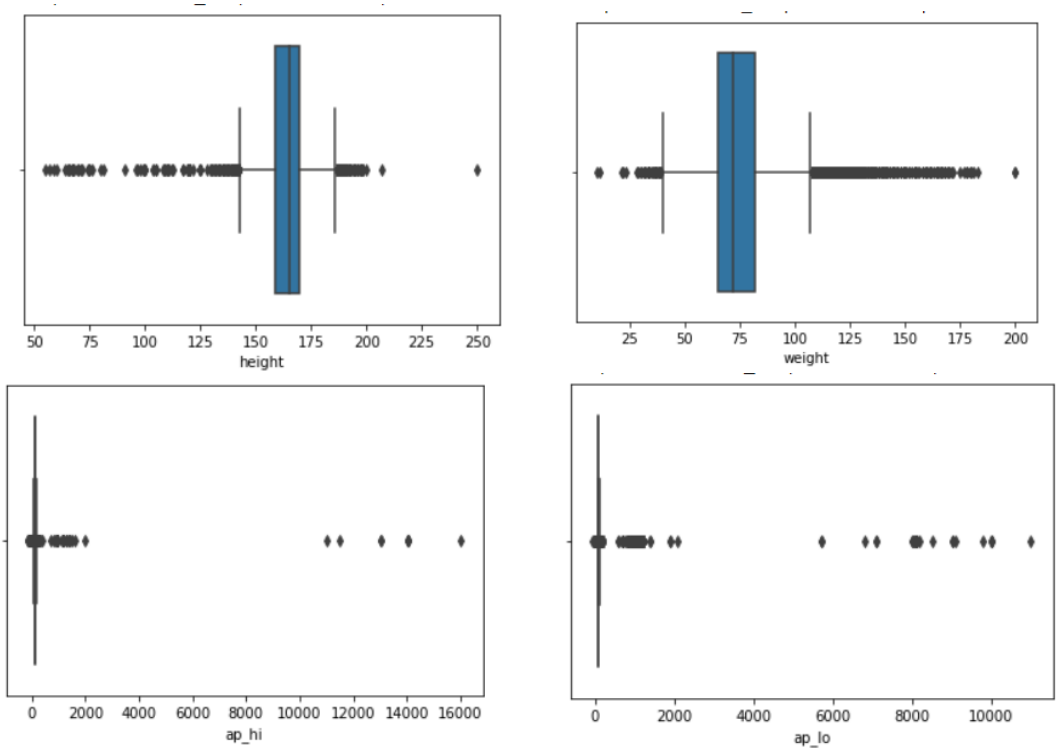

Count


Count
gluc


Count
active


Count
cholesterol


age_year

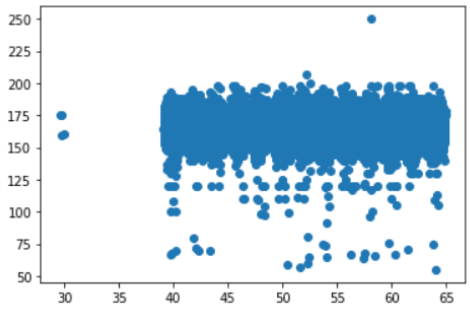


The graphs denoted are used to explain the presence of some datapoints which are outliers for the specific feature. But as they are part of some observation (data row), there is a probability that other features of that observation are making a noticeable contribution to the model. Hence, removing them directly is not a good practice. We will be removing outlier using the z-score test.

1. **Age VS Height:** The age variable in the dataset has values from 40 to 65. The height for that age group will fall in a specific range. However, there are some datapoint which are away from that range, these are the medical exceptions.
2. **Age VS Weight:** The age variable in the dataset has values from 40 to 65. The weight of a patient for that age group will fall in a specific range. However, there are some datapoint which are away from that range, these are the medical exceptions.
3. **Age VS Ap_hi:** The age variable in the dataset has values from 40 to 65. The high blood pressure value for that age group of patients is predefined. However, there are some datapoint which fall far away from that range, these are the medical exceptions.
4. **Age VS Ap_lo:** The age variable in the dataset has values from 40 to 65. The low blood pressure value for that age group of patients is predefined. However, there are some datapoint which fall far away from that range, these are the medical exceptions.

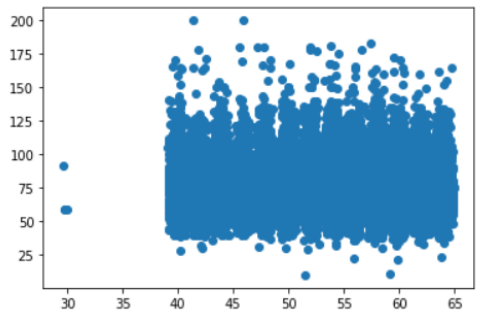
```
plt.scatter('age_year', 'height', data=mydata, marker='o')
```

```
<matplotlib.collections.PathCollection at 0x7f8cce682978>
```



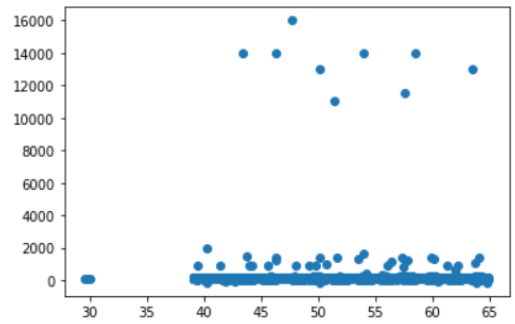
```
plt.scatter('age_year', 'weight', data=mydata, marker='o')
```

```
<matplotlib.collections.PathCollection at 0x7f8cce5f6b00>
```



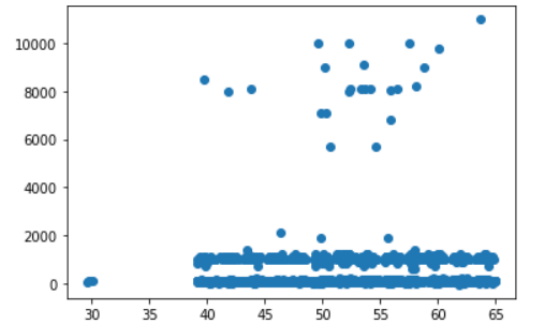
```
plt.scatter('age_year', 'ap_hi', data=mydata) #2000
```

```
<matplotlib.collections.PathCollection at 0x7f8cce5
```



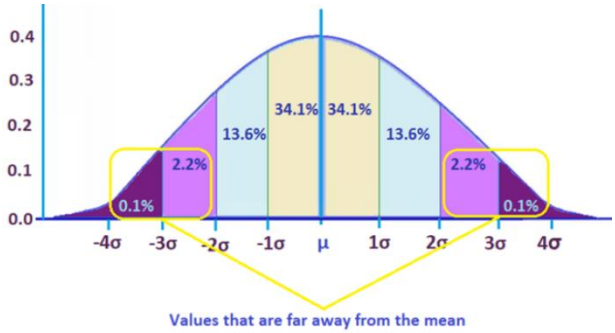
```
plt.scatter('age_year', 'ap_lo', data=mydata) #2000
```

```
<matplotlib.collections.PathCollection at 0x7f8cce5
```



Outlier Detection:

From analysis from above charts, we have concluded that our dataset has a few outliers. Hence to remove these outliers we will be using the **Z-score** outlier detection method. This score helps to understand if a data value is greater or smaller than data mean and how far away it is from the mean of the data. The z-score tells us how many standard deviations far the data point is from the mean.



The diagram represents that 68% of the data points of the dataset lies between +1 and -1 standard deviation. Also 95% of the data points lie between +2 and -2 standard deviation 99.7% of the data points lie between +3 and -3 standard deviation. To determine if the datapoint is away from other we find its z-score value. If the value of the z-score is more than 3 then the data point is an outlier.

According to the code below, the z-score of the dataset is calculated. The threshold for the value of z-score id defined as 3. We select all the datapoints in the data having z-score value less than 3.

```
[38] from scipy import stats
     import numpy as np
     z = np.abs(stats.zscore(mydata))
     print(z)
```

```
[[0.43654368 1.36435852 0.4532821  ... 0.23827041 0.49431072 0.99914126]
 [0.30734548 0.73294518 1.062197   ... 0.23827041 0.49431072 1.00085948]
 [0.2484431  0.73294518 0.07441232 ... 0.23827041 2.02301905 1.00085948]
 ...
 [0.16371676 1.36435852 2.34763097 ... 4.19691226 2.02301905 1.00085948]
 [1.20041787 0.73294518 0.17816753 ... 0.23827041 2.02301905 1.00085948]
 [0.43382691 0.73294518 0.70586195 ... 0.23827041 0.49431072 0.99914126]]
```

```
[39] threshold = 3
     print(np.where(z > 3))
```

```
(array([    7,    14,    14, ..., 69835, 69836, 69837]), array([7, 8, 9, ..., 8, 3, 9]))
```

```
[40] mydata1 = mydata[(z < 3).all(axis=1)]
```

```
[41] print(mydata.shape)
     print(mydata1.shape)
```

```
(69840, 12)
(55244, 12)
```

Data Preprocessing:

To make the model prediction even more accurate. The data must be scaled. Scaling of the data is nothing but the normalization of the data. The data need to be normalized a there as some features which have larger values and some have smaller values. The data values are brought in a specific range, just so that the weightage of each data value will be equal and no column of dataset will have more impact on the classification decision. Normalization is performed only on the numerical columns. We are using StandardScalar() function.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
```

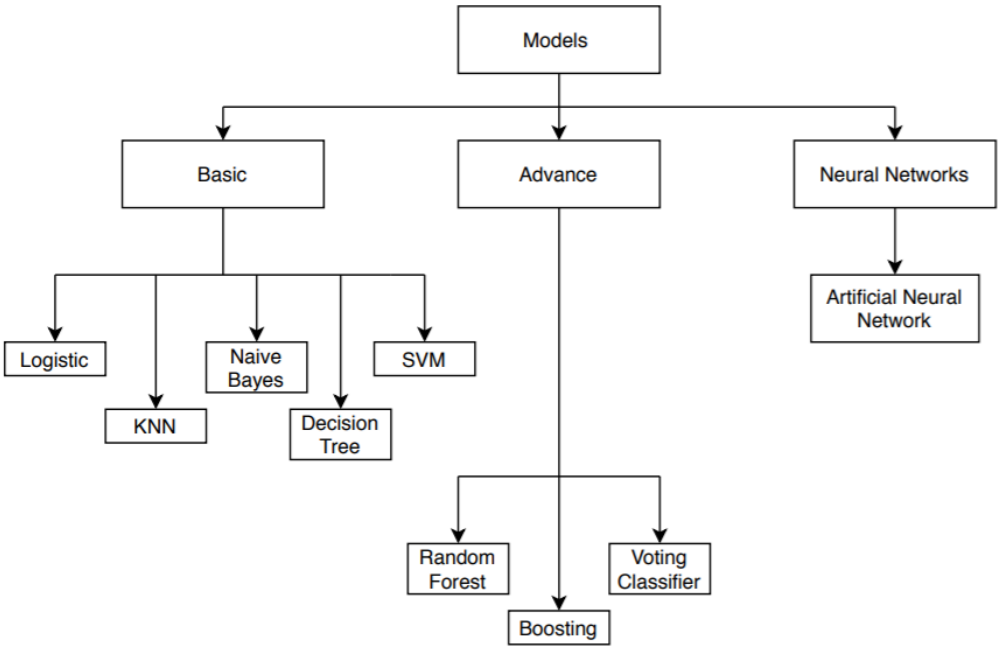
Data Splitting:

Data splitting means to divide the data into two parts, one portion of the data is used to describe the predictive model and the other is used for the model performance. Here, for building the model we need to split the data into test data and train data sets. Here, the splitting is being performed according to 70% and 30%. The screenshot represents the code for splitting of the data as 70- 30%. The model will be trained on the train dataset and the testing of the model performance will be on the test dataset.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 1)
```

Building Classification Models:

To find a solution to the given problem statement we are proposing multiple classification models. The problem statement requires the models to label each data entry into a class of either patient suffering from cardiac disease and patient without cardiac diseases. This classification models will be built on the training data set. The training dataset is 70% and the test data is 30%. The data set used here is scaled to give better performance of operation for each model.
We have built the following models:



1. Logistic Regression:

The logistic regression is the basic model which uses the logistic function which is used to build the model. The logistic regression is basically derived from the field of statistics. The logistic regression is also known as sigmoid function. The goal of logistic regression is to find the best fitting model.

The following code describes the logistic model we have built. The model is been made to find the probability of getting the cardio disease according to the features like body weight, cholesterol level, daily activity etc. According to the code implemented the model is been build.

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 1)
classifier.fit(X_train, y_train)
```

The next step after building the model the test data is been predicted.

```
y_pred = classifier.predict(X_test)
print(y_pred)
print(y_test)
#print(np.concatenate((y_pred, y_test),1))
```

The accuracy, Precision and recall is been found and further classification report is being determined.

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

The confusion matrix and the values for the accuracy, precision and recall are shown below.

```
[[8102 2380]
 [3366 7104]]
Accuracy: 0.725754104620084
Precision: 0.7490510333192746
Recall: 0.6785100286532951
```

The further evaluation of the model according to each label is being performed. The model here gives more precision at label 1 as compared to label 0. The score found here describes with good precision for each label and can be found that the data is not completely biased.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.71	0.77	0.74	10482
1	0.75	0.68	0.71	10470

2. KNN

The KNN classification the distance from the target variable is being calculated and is arranged according to minimum to maximum value. K determines the number of neighbors used for the classification of the target, depending on the number of nearest neighbors the label is being decided. Here, mainly the two ways are used to find the distance between the instance and the target, the Manhattan distance and the Euclidean distance. The final is selected according to the majority of label for the neighbors.

The K- nearest neighbor model classifies the data point according to the neighbor classified. Here we have used the Minkowski method to calculate the distance. The minkowski distance is the vector which is the generalized form of Manhattan distance and Euclidean distance. N_neighbors is the value of nearest neighbors to be considered. The code shows building the model.

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

Further the test data is been predicted after building the model.

```
y_pred = classifier.predict(X_test)
print(y_pred)
print(y_test)
#print(np.concatenate((y_pred, y_test),1))
```

The accuracy, Precision and recall is been found and further classification report is being determined.

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

The value of precision, accuracy, recall and the confusion matrix is been shown.

```
[[7280 3202]
 [3276 7194]]
Accuracy: 0.6908171057655593
Precision: 0.6919969218930357
Recall: 0.6871060171919771
```

The further evaluation of the model according to each label is being performed. The model here gives same precision at label 1 and label 0. The score found here describes with good precision for each label and is same so it can be found that the data is not at all biased.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.69	0.69	0.69	10482
1	0.69	0.69	0.69	10470

3. Naïve Bayes

Naïve bayes is the classification technique described on the basis of Bayes theorem showing the independence in between the predictors. The naïve bayes classifier shows the presence of the feature which are unrelated to the other features. This is easy classification algorithm and is used for larger data sets, also the naïve bayes is described to outperform the highly sophisticated methods of classification. The naïve bayes works better in multi class prediction and is fast to be performed.
The implementation code of Naïve Bayes is shown below. The naïve bayes requires less training data set and hence it is quicker than models like logistic regression.

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

The prediction of test data is been done after building of the model.

```
y_pred = classifier.predict(X_test)
print(y_pred)
print(y_test)
#print(np.concatenate((y_pred, y_test),1))
```

The classification report and the accuracy, precision and recall is found according to the following implementation

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

The value of accuracy, precision and recall is found to be:

```
[[9111 1371]
 [5784 4686]]
Accuracy: 0.6585051546391752
Precision: 0.7736503219415553
Recall: 0.44756446991404014
```

The further evaluation of the model according to each label is being performed. The model here gives more precision at label 1 as compared to label 0. The score found here does not describes with good precision for each label and it can be found that the difference is high hence it can be said that data is biased towards predicting label 1 as compared to label 0.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.61	0.87	0.72	10482
1	0.77	0.45	0.57	10470

4. Decision Tree

The decision tree is a part of supervised learning algorithms and it can be used to solve regression as well classification problems. The decision tree is created which can be used to predict the value of target variable. The decision tree is started from the root node and the further branches are created. In decision tree each node in the tree is described as the test case. This entire process is recursive and repeats after every new node is generated.
The model here is being implemented on the basis of entropy. The information gain is calculated on the criterion of entropy for the decision tree.

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 1)
classifier.fit(X_train, y_train)
```

The prediction of test data is performed after building of the model.

```
y_pred = classifier.predict(X_test)
print(y_pred)
print(y_test)
#print(np.concatenate((y_pred, y_test),1))
```

The classification report and the accuracy, precision and recall are found according to the following implementation

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

The value of accuracy, precision and recall is found to be:

```
[[6599 3883]
 [3736 6734]]
Accuracy: 0.6363592974417717
Precision: 0.634265800131864
Recall: 0.643170964660936
```

The further evaluation of the model according to each label is being performed. The model here gives more precision at label 0 as compared to label 1. The score found here describes with good precision for each label and can be found that the data is not completely biased.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.64	0.63	0.63	10482
1	0.63	0.64	0.64	10470

5. Support Vector Machine

Support vector machine (SVM) is an algorithm that is used for classification as well as regression, but generally it is used for classification. In SVM, the plot is created in n-dimensional space with the values that have particular coordinate. The support vectors are the points which are close to the hyperplane and affects the orientation of the hyperplane. The support vectors are used to increase the margin of the classifier.
The implementation of Support vector machine is been done by using the rbf function which creates a non-linear combination of the features used to uplift the sample into high dimensional feature space.

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 1)
classifier.fit(X_train, y_train)
```

The prediction of test data is performed after building of the model.

```
y_pred = classifier.predict(X_test)
print(y_pred)
print(y_test)
#print(np.concatenate((y_pred, y_test),1))
```

The classification report and the accuracy, precision and recall are found according to the following implementation

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

The value of accuracy, precision and recall is found to be:

```
[[8323 2159]
 [3491 6979]]
Accuracy: 0.730336006109202
Precision: 0.7637338586123879
Recall: 0.6665711556829035
```

The further evaluation of the model according to each label is being performed. The model here gives more precision at label 1 as compared to label 0. The score found here does not describes with good precision for each label and it can be found that the data is slightly biased.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.70	0.79	0.75	10482
1	0.76	0.67	0.71	10470

6. Random Forest

Random forest is a supervised learning algorithm and can be used without hyper-plane to produce the result. In this the decision tress are usually trained with bagging methods. The random forest searches the best feature among the random subsets of all the features instead of searching the feature while splitting of node and the random features are selected to split the node.

According to the code the random forest model is build. There are 10 estimators present. The information gain for the model is calculated based on the entropy criterion. Here the random state is set as 1 so that the data gets randomly selected.

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 1)
classifier.fit(X_train, y_train)
```

The next step after building the test data needs to be predicted

```
y_pred = classifier.predict(X_test)
print(y_pred)
print(y_test)
#print(np.concatenate((y_pred, y_test),1))
```

The overall accuracy of the matrix is given by the following code. The Calculation of the accuracy is not the criterion for the valuation. The precision and the recall values are also calculated for the model.

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

The confusion matrix and the values for the accuracy, precision and recall are given below.

```
[[7698 2784]
 [3626 6844]]
Accuracy: 0.6940626193203513
Precision: 0.7108433734939759
Recall: 0.6536771728748806
```

The evaluation of the model per label is also done using the following code. The model is predicting label 1 with more precision than the label 0. The score shown below show that the model predicts each label with good precision and is not completely biased.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.68	0.73	0.71	10482
1	0.71	0.65	0.68	10470

7. XGBoosting

Boosting algorithm improves the prediction power of any model by compensating the weakness of the predecessors and training the sequence of the weak models. We have implemented the XG boosting which is the further implementation of gradient boosting which is basically designed for improvising the speed and performance. Of the gradient descent method. It uses the same technique of gradient boosting. Here the weights are not incremented for misclassifications. The algorithm optimizes loss function of previous learners by adding a new adaptive model.
According to the code, we have fit the model on the train data.

```
from xgboost import XGBClassifier
classifier = XGBClassifier()
classifier.fit(X_train, y_train)
```

The model is predicting the test data values and a confusion matrix is built to evaluate the model. The precision, recall scores are also generated.

```
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

The following confusion matrix is generated along with the other evaluation values.

```
[[8139 2343]
 [3177 7293]]
Accuracy: 0.736540664375716
Precision: 0.7568493150684932
Recall: 0.6965616045845272
```

The evaluation of the model per label is also done using the following code. The model is predicting label 1 with more precision than the label 0. The score shown below show that the model predicts each label with good precision and is not completely biased.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.72	0.78	0.75	10482
1	0.76	0.70	0.73	10470

8. Voting Classifier

Voting classifier is the machine learning algorithm which trains on ensemble of various models and predicts the output by determining the highest probability. The voting classifier generally supports two type of voting, hard voting and soft voting. The hard voting provides the output with the highest probability whereas the soft voting provides with the average probability. The model is a part of ensemble learning. In this algorithm multiple ML models are built. Initially we train different models and make individual predictions. Then the Voting Classifier uses majority votes to make predictions.

We have used 3 models: KNN, Logistic and Random Forest. The type of voting used here is hard voting, this type gives us the output as either 0 or 1. We initially had built the models individually and calculated its accuracy for that and then built it using the voting classifier and calculated the accuracy for it.

```
#test the three models with the test data and print their accuracy scores
print('knn: {}'.format(knn_best.score(X_test, y_test)))
print('rf: {}'.format(rf_best.score(X_test, y_test)))
print('log_reg: {}'.format(log_reg.score(X_test, y_test)))

#Building the Voting Classifier
from sklearn.ensemble import VotingClassifier
#create a dictionary of our models
estimators=[('knn', knn_best), ('rf', rf_best), ('log_reg', log_reg)]
#create our voting classifier, inputting our models
ensemble = VotingClassifier(estimators, voting='hard')

#fit model to training data
ensemble.fit(X_train, y_train)
#test our model on the test data
ensemble.score(X_test, y_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
[[8131 2351]
 [3371 7099]]
Accuracy: 0.7268995799923635
Precision: 0.7512169312169312
Recall: 0.6780324737344795
```

According to the screenshot, the accuracy values for the individual models has been improved when all the algorithms are used together.

```
{'n_neighbors': 24}
{'n_estimators': 200}
knn: 0.7219358533791523
rf: 0.7051355479190531
log_reg: 0.7250381825124094
0.7264700267277587
```

The evaluation of the model per label is also done using the following code. The model is predicting label 1 with more precision than the label 0. The score shown below show that the model predicts each label with good precision and is not completely biased.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.71	0.78	0.74	10482
1	0.75	0.68	0.71	10470

9. Artificial Neural Network

Artificial neural network (ANN) is applied to more complex patterns and prediction problems. The structure of ANN also contains a hidden layer which is called distillation layer, this layer is used as it make the network faster and more efficient because it uses the important information and leaves the redundant information. The ANN neural network contains the ability to learn and model the complex and non-linear relations. Trains themselves to recognize pattern in the data and predict output for similar data.

According to the code we have used the TensorFlow library to build the neural network. We have created a sequential layer and then added 4 dense layers (hidden layers). The number of neurons sin the layers are 32 each. The activation function used here is RELU. This function will decide the output for the given input. RELU function gives 0 for negative values and considers the positive values.

```
import tensorflow.keras
from keras.models import Sequential
from keras.layers import Dense, MaxPooling2D, Flatten, Dropout, BatchNormalization, Conv2D
from sklearn.preprocessing import MinMaxScaler
from keras.optimizers import RMSprop, SGD, Adam
```

```
model = Sequential()    #creating a sequential model
model.add(Dense(32, activation = "relu",input_shape=(52380, 11)) )
model.add(Dense(32, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='relu'))

model.summary()
```

The optimizer used here is the Adam optimizer. With the help of optimizer, we can change the weight of a neuron for further calculations. Loss function is used as mean squared error has to be minimized for the model. An epoch refers to one cycle through the full training dataset. Here the epoch is 100. Batch size of the data is the batches in which the data will go in the set of batches, here 64.

```
opt = Adam(lr=0.01)
model.compile(optimizer=opt, loss='mean_squared_error',metrics=['accuracy'])  #using adam optimizer to update the
                                                                               #weights and mean squared error is to be reduced
```

```
epochs_hist = model.fit(X_train, y_train, epochs=100, batch_size=64, validation_split=0.2)
#the traing of the model is done for 100 iteration
```

```
612/612 [==============================] - 1s 2ms/step - loss: 0.1789 - accuracy: 0.7391 - val_loss: 0.1815 - val_accuracy: 0.7330
Epoch 96/100
612/612 [==============================] - 1s 2ms/step - loss: 0.1791 - accuracy: 0.7379 - val_loss: 0.1809 - val_accuracy: 0.7344
Epoch 97/100
612/612 [==============================] - 1s 2ms/step - loss: 0.1789 - accuracy: 0.7386 - val_loss: 0.1814 - val_accuracy: 0.7297
Epoch 98/100
612/612 [==============================] - 1s 2ms/step - loss: 0.1788 - accuracy: 0.7394 - val_loss: 0.1838 - val_accuracy: 0.7319
Epoch 99/100
612/612 [==============================] - 1s 2ms/step - loss: 0.1792 - accuracy: 0.7381 - val_loss: 0.1815 - val_accuracy: 0.7332
Epoch 100/100
612/612 [==============================] - 1s 2ms/step - loss: 0.1791 - accuracy: 0.7378 - val_loss: 0.1826 - val_accuracy: 0.7309
```

The above output shows that the model has worked for 100 epochs an dthe accuracy for each iteration is shown.

```
y_pred = model.predict(X_test)
y_pred = y_pred.round()
print(y_pred)
y_pred = np.argmax(y_pred, axis=1)
print(y_test)
y_test = np.argmax(y_test, axis=1)
```

```
WARNING:tensorflow:Model was constructed w
[[0. 1.]
 [0. 1.]
 [1. 0.]
 ...
 [1. 0.]
 [0. 1.]
 [0. 1.]]
[[1. 0.]
 [1. 0.]
 [0. 1.]
 ...
 [1. 0.]
 [1. 0.]
 [0. 1.]]
```

The confusion matrix is created as follows:

```
from sklearn.metrics import confusion_matrix, accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
#print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
#print("Precision:",metrics.precision_score(y_test, y_pred))
#print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
[[7822 2660]
 [2935 7535]]
0.7329610538373424
```

The evaluation of the model per label is also done using the following code. The model is predicting label 1 with more precision than the label 0. The score shown below show that the model predicts label 1 with a slightly better precision than the label 0 but this difference can be accepted and we can say that the model is not completely biased.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.70	0.79	0.74	10482
1	0.76	0.67	0.71	10470

5.2 Evaluations and Results

The model built above will be tested on the test data set. The evaluation of the model will be done based on the value of different **accuracy matrices** for the classification part. The accuracy matrices for different models will measure how accurate our model can predict the target variable. A model is expected to have high accuracy to be selected. Performance evaluation of the Models is done on the bases of confusion matrix. The important terms for predicting performance are:
True positives (TP): the cases for which the classifier predicted '1' and the value was actually 1.
True negatives (TN): the cases for which the classifier predicted '0' and the value was actually 0.
False positives (FP): the cases for which the classifier predicted '1' but the value was actually 0.
False negatives (FN): the cases for which the classifier predicted '0' but the value was actually 1

		Predicted class: *P*	Predicted class: *N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Performance Measures:

Precision: ratio between the True Positives and all the Positives. the measure of patients that we correctly identify having a cardio disease out of all the patients. The best model must be selected considering a high value of precision.

$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)}$$

Recall: Measure of model correctly identifying True Positives. For all the patients who actually have heart disease, recall tells how many we correctly identified as having a cardio disease. The best model must be selected considering a high value of recall.

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(FN)}$$

Best Model Selection:

Models	Accuracy	Precision	Recall
Logistic	72.57	74.90	67.85
KNN	69.08	69.19	68.71
Naive Byes	65.85	77.36	44.75
Decision tree	63.63	63.42	64.31
SVM	73.03	76.37	66.65
Random forest	69.40	71.08	65.36
XGBoost	73.65	75.68	69.65
Voting classifier	72.6	75.1	67.8
Artificial Neural Network	73.3	72.7	74.6

The table above shows the comparison of all the models on the biases of accuracy, precision and recall. There are three models Which have the best accuracy values. The models are SVM, XGBoost and Artificial Neural Network. The Precision value for predicting the results of these three models is almost the same. Hence the next criterion for selecting the best model is the recall value. There is a significant difference in the recall value of the models. The highest value is for the Artificial Neural Network.
Hence on the basis of comparing the performance measures of the models the best model is the **Artificial Neural Network.**

Evaluation per label:

Now we will evaluate the models on their performance measure of predicting labels correctly. The table given bellow shows us the values for per label prediction for each model. Based on the overall accuracy of the model we have selected the Artificial neural network, referring this table for the performance of prediction per label, the ANN model has a good precision.

Model	Label	Precision	Recall
Logistic	0	71	77
	1	75	68
KNN	0	69	69
	1	69	69
Naïve Bayes	0	61	87
	1	77	45
Decision Trees	0	64	63
	1	63	64
SVM	0	70	79
	1	76	67
Random Forest	0	68	73
	1	71	65
XGBoost	0	72	78
	1	76	70
Voting Classifier	0	71	78
	1	75	68
Artificial Neural Networks	0	70	79
	1	76	67

Hence after considering both combined prediction performance and per label performance measure we can conclude that the **Artificial Neural Network is best for this dataset**.

5.3 Findings

The main and the final conclusion is to find the best model for performing the prediction. The following models were and the accuracy, precision and recall for each of the model is been predicted. the comparison of accuracy for each model is done and the final and best model is predicted for the cardio disease data. The precision and recall are determined for each label (0 and 1). The artificial neural network gave the highest accuracy. From the confusion matrix output it was clear the number of misclassifications had reduced to a very small amount. The precision in artificial neural network was found to be 70 for label 0 and 76 for label 1. The recall in artificial neural network is 79 for label 0 and 67 for label 1. In the SVM, SVM kernel is being used. Finally, it is being concluded that artificial neural network makes the best model in all the model performed.

6. Expected Outcomes

6.1 Conclusions

The main motive to build this project was to be able to predict the values and find whether the person is prone to cardio disease or not according to the features or the attributes.
Whereas the classification models will be used for classifying the person will have cardio disease or not.
Here, according to the models build and the accuracy found we find that the artificial neural network is the best model. Also, the values for precision and recall were calculated for the label 0 and label 1 and were found to be satisfactory. The values of precision and recall for each of the model is determined. This value tells us how much accurate and reliable our model is, how much perfect its predictions will be. As a real-life project, it will not be dependable if the value of accuracy of prediction is low. This would mean that with every decreasing value of accuracy there is an increase in the value of mistake. After comparing the accuracy values for all the models built, it is concluded that the **Artificial neural network model is the best model for prediction for the cardio disease dataset**. This model is useful in forecasting the patient's state of health and to predict it beforehand.

6.2 Limitations

The limitation of this model is that there are other features that will affect whether the patient will suffer from cardio disease or not, these features are not present in the current dataset. Sometimes there are exceptions of patients having all positive signs of cardio disease but do not have it. This can be misleading and hence is responsible for reducing the accuracy of the model.

6.3 Potential Improvements or Future Work

The classified data can be clustered together into clusters of patients having the disease or not. For clustering we will be using K-Means Clustering, Mean shift Clustering, Centroid based Clustering. The visualization can be performed on the dataset to help better understand the pattern in the data and can be useful for instant insights into it.