Cardio-Vascular Risk Detection using different Machine Learning Algorithms

Badal Parmar
18BCP011
Computer Science Engineering
Pandit Deendayal Energy University
Gandhinagar, Gujarat
badal.pce18@sot.pdpu.ac.in

Rajeev Gupta
Computer Science Engineering
Pandit Deendayal Energy University
Gandhinagar, Gujarat
rajeev.gupta@sot.pdpu.ac.in

***Abstract* - cardiovascular disease is a prominent cause of disability and early death all over the world, and it is a major contributor of growing healthcare costs. Atherosclerosis is the underlying pathology, which develops over time and is usually advanced by the time symptoms appear, usually in middle life. Acute coronary and cerebrovascular crises occur suddenly, and patients might die before medical treatment arrives. Modifying risk factors has been shown to reduce mortality and morbidity in people with known or undiagnosed cardiovascular disease. Classification methods are used frequently in Medical Sciences. One of the major problems faced in Medical Science is Cardiovascular Risk Detection. Classification has been used intensively to group patients into Risky and Non-Risky. Risky class refers to the patients who are under the looming threat of Cardiovascular attack. The non-risky class has patients who are safe from Cardiovascular attacks. The various diseases to detect the risk as per research are Diabetes, IHD, Hypertension, and Arrhythmia. But not all classification methods can be accurate. Different Machine learning algorithms are used for rick detection for Classification. The Cardio-Vascular Legacy dataset has been utilized for experimentation. This work addresses how the general Machine Learning Algorithms can be utilized for building a Risk-Detection model. Extensive experimentation was performed, and the model achieved a recognition accuracy of 99%. The comparison of accuracies was done for different predictions recorded through various algorithms. The range of accuracies varied from 95% to 99%. The present approach attempts to determine if a patient is at danger or not. As work is done with a 'if or else' classifying, often known as a '0 or 1' classification mode (binary classifier), the model is binary. Early detection of Cardiovascular risk can prove to be highly useful for humankind. There are millions of deaths reported due to cardiovascular complications every year. The research aims to provide a patient-level risk detection system so that patients can know the risk beforehand and follow their prescribed prevention routines.**

Keywords – Cardio-Vascular Disease, Machine Learning, KNN, Decision Tree Classification, Neural Networks, .

1. INTRODUCTION

In 2015, 17.7 million individuals died worldwide as a result of cardiovascular diseases. By 2030, this number is expected to climb to 22 million. The increased frequency of non-communicable diseases (NCDs) is putting pressure on healthcare systems and businesses around the world, especially when combined with an aging population. Although the prevalence of various cardiovascular risk factors is growing at diverse rates throughout the world, the magnitude of the gain has prompted researchers to focus on the reasons of the risk variables (i.e., the "roots of the causes"), which includes socioeconomic aspects of health.

Cardiovascular disease is the biggest cause of death in the world today. It is believed that 3.8 million males and 3.4 million females are killed each year. Cardiovascular risk is linked to both diastolic and systolic blood pressure. As a result, a precise and feasible prognosis of heart-related disorders is critical. Medical institutes all across the globe gather data on a variety of health-related issues. Using a number of machine learning algorithms, this data may be leveraged to derive valuable insights. The data accumulated, however, is massive, and it is usually noisy. These data, which are too huge for humans to comprehend, might be simply processed using machine learning techniques. As a result, these approaches have lately been shown to be highly useful in accurately predicting the existence or absence of cardio illnesses. To begin, in our approach, we have been using a variety of strategies and procedures. In this research, the classification of Machine Learning techniques and algorithms is used to enhance the accuracy rate. In ML, classification algorithms or supervised learning strategies are utilized in which the system adapts from the raw training dataset. This research procedure might be bi-class or multi-class (for example, deciding if the individual is male or female or if the medicine is useful or not).
Here are the names of the categorization algorithms that are used in this project:

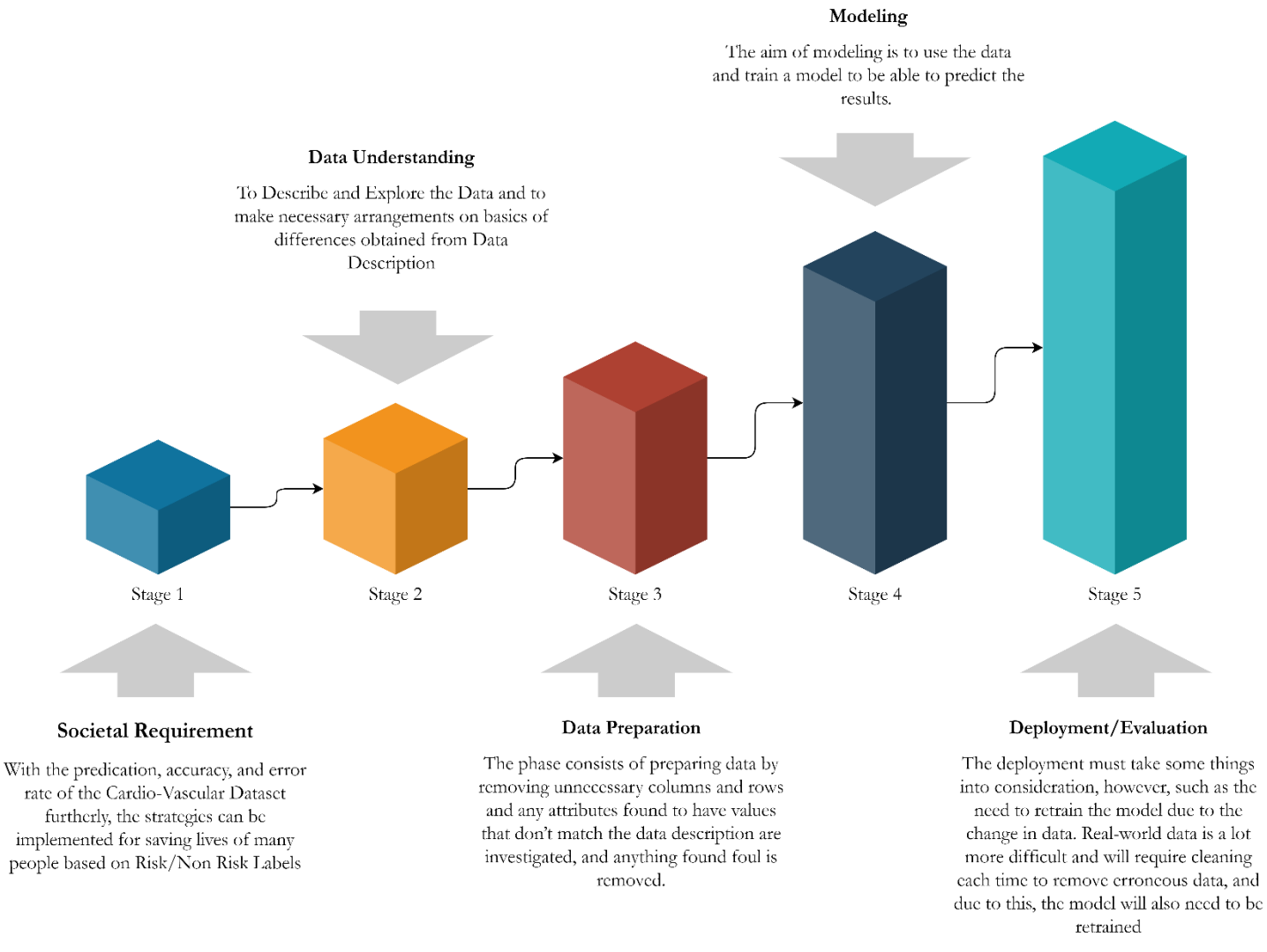
1) K Nearest Neighbour
2) Decision Tree Classification

2. ORGANIZATION OF PAPER

SECTION 1	SECTION 2	SECTION 3	SECTION 4	SECTION 5
MACHINE LEARNING FLOW	**ARCHITECTURE**	**EXPERIMENT**	**RESULTS**	**CONCLUSION**
WORKING METHODOLOGY	**DATA ANALYSIS**	**IMPLEMENTATION OF CLASSIFICATION ALGORITHMS**	**COMPARISION ANALYSIS**	**CONCLUSION AND FUTURE WORK**
The section consists of the Workflow of process and dataset briefing	This section discusses the framework, working, advantages, and possible use cases of our system. Contains the block diagrams of Systems processes and learning models	Description of processes and steps also contains the Google Collab Link for complete Juypter Notebook Code	Accuracies/Prediction results in comparison of Classification Algorithm	The final section concludes the project and provides an overview of how this proposed system could be integrated with in the future for further implementation.

3. WORKING PROCEDURE

3.1. WORKFLOW OF ML MODEL



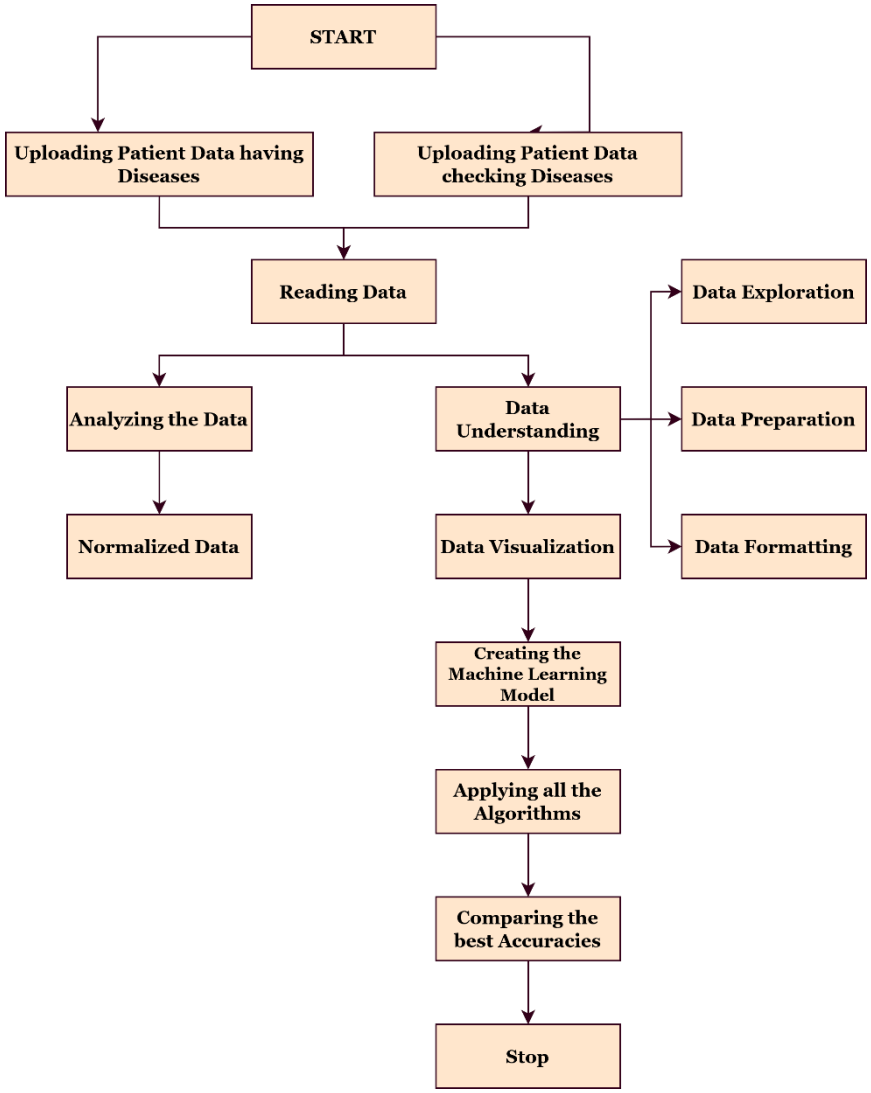
3.2. DATA ACQUISITION

→ Cardio-vascular disease dataset is extracted from Kaggle official website.

3.3. DATA PREPROCESSING

→ Data preparation is essential to develop a more accurate Machine Learning model. The process of cleansing data is known as data pre-processing. All NAN values will be removed from our data. Data Wrangling is another name for this procedure. During data pre-processing missing data, noisy data, and inconsistent data are identified.

3.4. BLOCK MODEL OF PROPOSED ARCHITECTURE



3.5. SELECTION OF MACHINE LEARNING MODEL

→ Machine learning methods are then used to identify the pre-processed data. Comparing the best accuracy among the Classification Algorithm.

3.5.1. INPUT VARIABLE FOR PREDICTIONS

→ **Dataset consists of 11 Rows. The Machine Learning methodology is focused on estimating label – Risk/No Risk.**

VARIABLE NAME	VARIABLE TYPE	VARIABLE COUNT
Random	Float64	1520
ID	Int64	1520
Indication	Object	1517
Diabetes	Object	1518
IHD	Object	1520
Hypertension	Object	1517
Arrhythmia	Object	1520
History	Object	1518
IPSI	Float64	1516
Contra	Object	1520
Label	Object	1517

4. DATA ANALYSIS

→ Python 3.8.8 was used to evaluate quantitative data in Juypter Notebook for further categorization.

4.1. IMPORTING THE LIBRARIES

→ Importing required libraries to create a machine learning model in Juypter Notebook.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn import preprocessing
from sklearn import metrics

from sklearn.utils.multiclass import unique_labels
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

# Import libraries for k-fold and cross validation,
#to test model accuracy using a mean score of k
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import LinearRegression
```

```
# Import libraries from sklearn to create the models used for classification
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# Import libraries to be used for accuracy score,
# classification report in form of a confusion matrix
# and the code to find the mean absolute error of the model
from sklearn.metrics import accuracy_score
from yellowbrick.classifier import ClassificationReport
from yellowbrick.classifier import DiscriminationThreshold
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# Import libraries for decision tree creation
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
```

4.2. READING THE DATASET

	Random	Id	Indication	Diabetes	IHD	Hypertension	Arrhythmia	History	IPSI	Contra	label
0	0.602437	218242	A-F	no	no	yes	no	no	78.0	20	NoRisk
1	0.602437	159284	TIA	no	no	no	no	no	70.0	60	NoRisk
2	0.602437	106066	A-F	no	yes	yes	no	no	95.0	40	Risk
3	0.128157	229592	TIA	no	no	yes	no	no	90.0	85	Risk
4	0.676862	245829	CVA	no	no	no	no	no	70.0	20	NoRisk
...	...	...	...	...	...	...	...	...	...	...	...
1515	0.391440	93406	A-F	no	yes	no	no	no	76.0	60	NoRisk
1516	0.253504	121814	A-F	no	no	yes	yes	no	90.0	75	Risk
1517	0.620373	101754	TIA	no	no	yes	no	no	75.0	20	NoRisk
1518	0.639342	263836	A-F	no	yes	no	no	no	70.0	45	NoRisk
1519	0.634922	254941	CVA	no	no	no	no	no	60.0	20	NoRisk

1520 rows × 11 columns

4.3. DATA PRE-PROCESSING

→ For data preprocessing, the frequency of null values for cleaning an existing dataset has been evaluated. The following table illustrates the total missing parameters based on the features.

```
Indication      0.197368
Diabetes        0.131579
IHD             0.000000
Hypertension    0.197368
Arrhythmia      0.000000
History         0.131579
IPSI            0.263158
Contra          0.065789
label           0.328947
dtype: float64
```

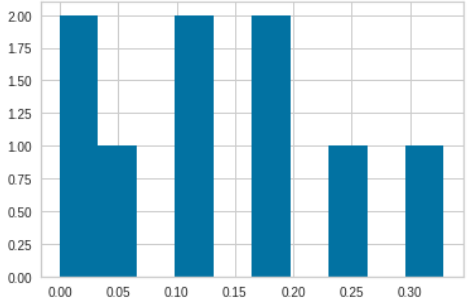


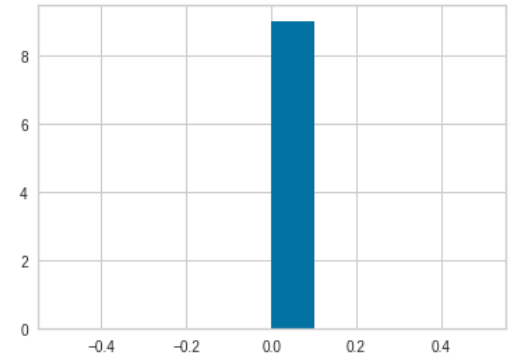
Fig.: Percentage of missing/null values in the Dataset

→ The Pandas Data Frame was used to figure out what proportion of the column's values were missing. The total number of rows with missing values are 20, however because they account for 1.31 percent of the overall dataset, they are discarded. The dropna () function in Pandas was used to examine the Null values in the columns and rows.

```
# Drop all data values that are null
clean_data = clean_data.dropna()
```

→ **The drop rows/columns are shown in the graph which is the results of using the Pandas dropna ()**

```
Indication      0
Diabetes        0
IHD             0
Hypertension    0
Arrhythmia      0
History         0
IPSI            0
Contra          0
label           0
dtype: int64
```



→ **The Label (Risk/No Risk) of Cardio-vascular disease dataset has shown in figure below.**

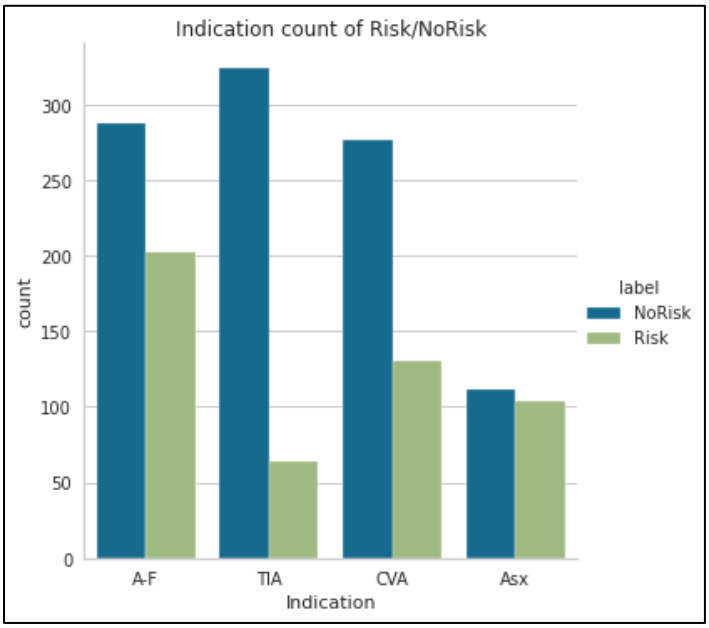
```
# Run a check to make sure the label column only has risk and no risk
clean_data["label"].value_counts()
```

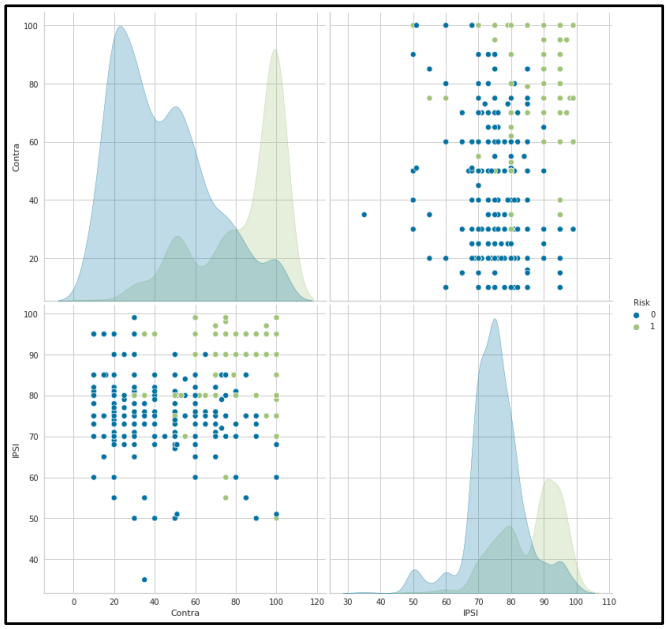
```
NoRisk    1000
Risk       500
Name: label, dtype: int64
```

→ **As shown in above data, there are 1000 patients with No Risk and 500 patients who are at risk.**

4.4. DATA VISUALIZATION

4.4.1. Visualization using BarPlot, Pairplot & Correlation



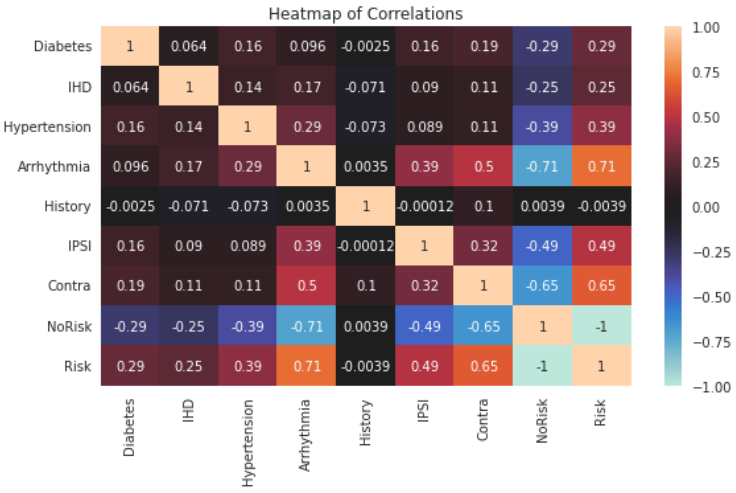


Following are the visualization created using JupyterLab

[1] TIA: More with No Risk, less Risk
[2] CVAL More with No Risk, bit with Risk
[3] AF: More with No Risk, More with Risk
[4] Asx: Equal Risk/No Risk

PairPlot between Contra and IPSI for the estimation of Risk and No Risk Classification in Label Column. Because upper table shows how the patients with Risk have much higher value of Contra but not much with IPSI. On other hand most No Risk Patient have a lower comparison mean value.
It's clear from Graph with the increase in value of Contra there is increasing Risk of Patients while it was observed that IPSI was not found that important.

4.4.2. Visualization using Correlation Matrix

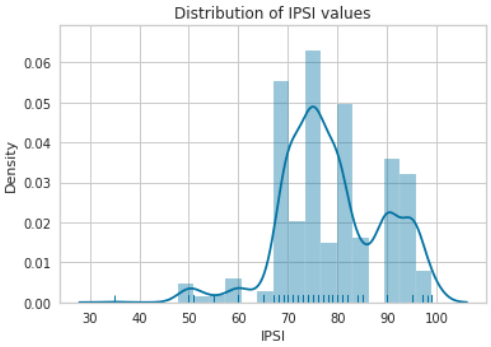


⇨ **Correlation Matrix Analysis:** With the increase in Contra value the Patients' risk value increases - Contra has + 0.65 correlation with Risk and -0.65 correlation with No Risk.

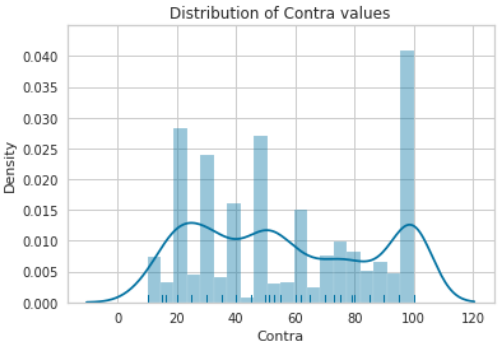
→ IPSI has +0.49 and -0.49 score shows it's not as impactful value as CONTRA

→ The most interesting Correlation score from the heatmap however, is Arrhythmia. It has a 0.71 for Risk and -0.71 for No Risk. Taking into consideration what this attribute represents, it makes sense that it has the highest correlation with the outcome.

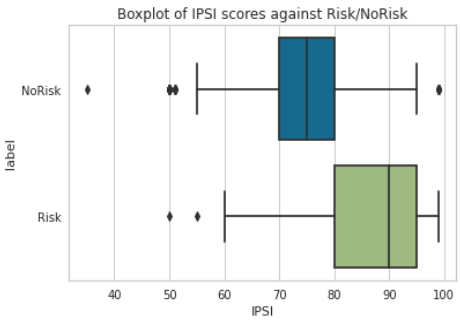
4.4.3. Distribution of IPSI and Contra across the Dataset

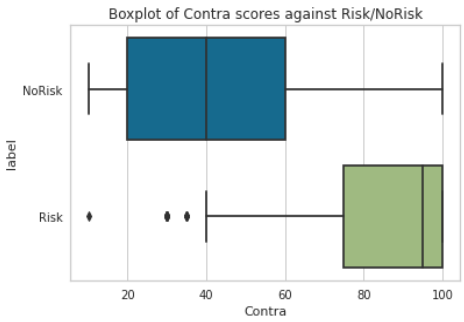


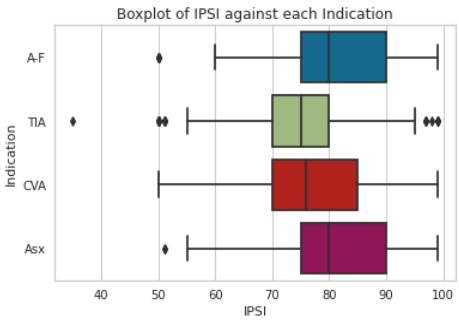
The distribution graphs help to give a visual understanding of where most of the values lie in the dataset. The IPSI data tends to fall mostly in the 60-80 value area, as indicated by the distribution line, whilst the Contra tends to have a more equal variation of values that flow across the entire dataset, with the highest values seemingly appearing at a 100 value, the highest possible value.



4.4.4. Visualization using BOXPLOT



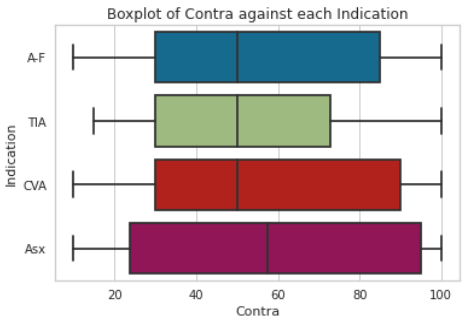




From the IPSI boxplot below, the majority of the IPSI values for norisk are below 80, with a few outliers nearer the bottom of the boxplot, and a single outlier located at the value of 99. Whilst the risk patients generally have a score above 80, with the median here of 89. A few outliers are present for the risk patients, who have an IPSI value between 50 and 60.

From the Contra boxplot, the picture is much clearer. The majority of no-risk patients have a Contra value below 60 and no visible outliers. The risk patients are mostly above 75, with the median here of around 96, a very high value, with a few outliers below 40. The majority of risk patients maintain high Contra scores, with a lower distribution being below a value of 40.

While IPSI having a high value doesn't necessarily mean risk, having a high Contra score means that the majority of the time, it will result in a classification of risk. The IPSI values generally seem to be more distributed b/w 70 and 100 across all 4 indication classes. There are a few outliers between the different classes, but none seem to be substantial here.



The boxplot against the Indication class sees no real difference here, as the values are mostly distributed in a way that doesn't give a clear insight as to whether the Indication class has an effect on the end classification. The boxplot for the Contra however is a little bit more interesting. Whilst the boxplot for IPSI toward Indication class is mostly as expected due to the right skewing of the data, the Contra boxplot is more interesting. This is because the wide variety of Contra values range across all classes and shows no outliers at all.

The most important thing to get from this boxplot, is the Asx class, and TIA class. The Asx class has a very high spread of data within its interquartile range, whilst TIA is more evenly spread across the middle of the dataset. We can also see that 3 of the 4 classes have an exact median score of around 50, whilst Asx has a slightly higher median of around 58.

4.5. NORMALIZATION OF DATA VALUES

→ Data formatting is required to guarantee that the data is in the correct format for later usage by models. Normalization and One-hot Encoding were the two data formatting techniques that were employed. Normalization, also known as feature scaling, is the procedure for converting all numerical values to a range of 0 to 1. This is done so that the values can be converted to a more comparable basis while maintaining their actual values (Data Normalization section in Jupyter Notebook). Categorical values in the dataset must be pre-processed, which is done using the label encoder function in the sklearn library. This is done to ensure that all categorical values are numerically equivalent. Many of the data points in the dataset are categorical (yes/no), which will be converted to 0 and 1 to create a more common scale that machine learning models can analyse and visualize in graphs.

4.6. TRAINING AND TESTING THE DATASET

→ The original data was separated into training and testing sets for the evaluation phase. A sci-kit learn library was utilized.

→ The figure below shows that the x and y variables were declared. After this is done, the data will be split into a 90/10 training and testing split. Then the shapes will be checked to make sure this was done correctly. An unshuffled split was done, as the plan was to test the models through cross-validation afterward.

→ Null accuracy was initially defined. Each model must always get a higher score than the null value listed above, or it will be labelled as a dumb model.

```
[ ]  #declare the target of the model
     target = clean_data_process['label']

[ ]  #create and split up the training and test data for the models
     x = meddata
     y = target

     #create the train and test sets
     x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.1, shuffle=False)

     print(x_train.shape)
     print(y_train.shape)
     print(x_test.shape)
     print(y_test.shape)

     (1350, 11)
     (1350,)
     (150, 11)
     (150,)
```

```
# examine class distribution
print(y_test.value_counts(), "\n")

# calculate the percentage of 0
print(1 - y_test.mean(), "\n")

# calculate the null accuracy, in a single line of code
print("%0.2f " % max(y_test.mean(), 1 - y_test.mean()))

0    104
1     46
Name: label, dtype: int64

0.6933333333333334

0.69
```

5. IMPLEMENTATION OF CLASSIFICATION ALGORITHMS

GOOGLE COLLAB LINK:
https://colab.research.google.com/drive/1Y756_WeN6Mfy7leelnNYBKIgR0i6oXOj?usp=sharing

DATASET LINK:
https://drive.google.com/file/d/1OIyTZ24ng0n6XWmGav2IFLG8DE4AER5i/view?usp=sharing

→ The inputs and outputs are the same across all models. The train/test split approach is used to train each model, and then cross-validation is used for additional training. Each model has been trained on X and then tested on Y, with X being the data and Y representing the target. To ensure that the accuracy attained is sufficient, the precision metrics were assessed using cross-validation at 50 folds. The accuracy is then examined using a train/test split. Cross-validation focuses on determining whether the dataset split is even and well reflective of the entire data we're using, as well as the model is not suffering from overfitting. Many iterations are used to train models. The models employed didn't contain much configurable hyper-parameters, but when they did, multiple techniques were applied and the best outcome was preserved, and thereafter cross-validation was performed using the same hyper-parameters. The learning rate, the solver the model will employ, and the maximum number of iterations the simulation should run are all hyper-parameters. The maximum depth of a decision tree can be modified. Confusion Matrices and Classification Reports were used as the metrics for determining how well the models performed and which one seems to be the viable choice among all the possibilities.

→ A classification report and associated confusion matrix are used to evaluate models. While the code displays not just the outcome, but also the hyper-parameters that were tested so that the prediction accuracy can be compared.

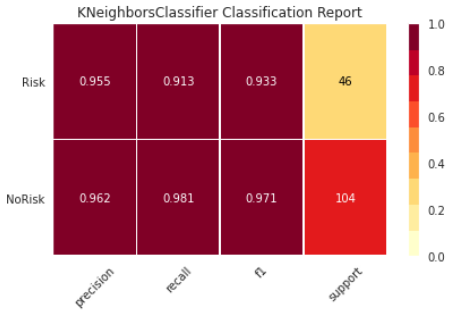
5.1. KNN CLASSIFICATION

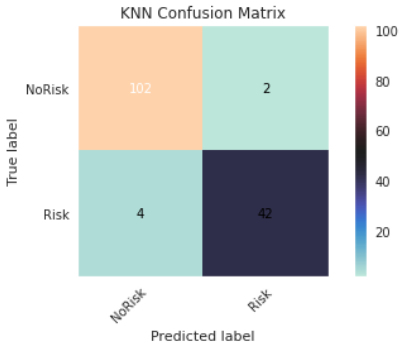
→ KNN or K-Nearest-Neighbours is a simple and easy implement algorithm. This algorithm assumes that similar things exist nearby, which we define as K. It is possible to achieve varying results of accuracy by doing something as simple as changing the sum of K for the model. As the data in the train/test split totals 150, 2 seems like the best k to start with.

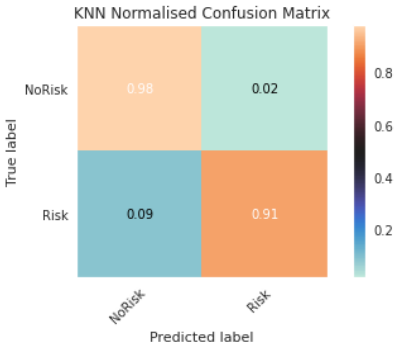
→ **KNN CLASSIFICATION FOR K=2**

```
neigh = KNeighborsClassifier(n_neighbors=2)
neigh.fit(x_train, y_train)
# predict the response
pred = neigh.predict(x_test)
# evaluate accuracy
knnaccuracy = accuracy_score(y_test, pred, normalize = True)
print ("KNeighbors K=2 classification accuracy score : %0.2f " % knnaccuracy)

KNeighbors K=2 classification accuracy score : 0.96
```

In the case of K = 2, the model achieves a good accuracy. The classification report and confusion matrix are shown below, to get a better visual idea of how the results are spread out







→ The normalized Confusion matrix shows a 98% precision for No Risk classification and a 91% precision for Risk classification. Furtherly, cross-validation on KNN is run to see how it differs from the train/test split.

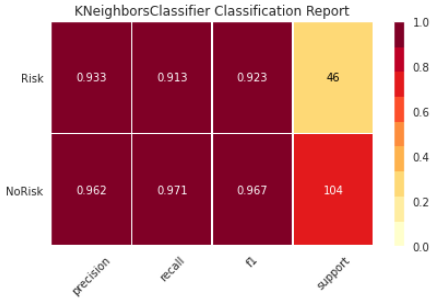
```
neigh = KNeighborsClassifier(n_neighbors=2)
#Train the algorithm
knnscore = cross_val_score(neigh, x, y, cv=50, scoring='accuracy')
print(knnscore)
print("\n")
print("KNeighbours Cross-Validation Accuracy : %0.2f (+/- %0.2f)" % \
      (knnscore.mean(), knnscore.std() * 2))

[1.   1.   1.   1.   1.   1.   0.97 1.   0.97 1.   1.   0.97 1.   0.97
 0.97 1.   1.   1.   1.   0.93 1.   0.97 1.   0.97 1.   1.   1.   1.
 1.   0.97 0.97 1.   1.   0.97 0.97 1.   1.   1.   0.93 1.   1.   1.
 0.97 1.   0.97 1.   1.   1.   0.97 0.97]


KNeighbours Cross-Validation Accuracy : 0.99 (+/- 0.04)
```

Cross-Validation achieves a mean accuracy of 99% with a variance of 0.04. The scores themselves stay near 100% except for a couple of scores at 93% and 97%, which decreases the mean to 99%.
The K value can have an impact on what sort of accuracy the model achieves, due to the way KNN works. Let's run a comparison and visualize it within a graph, to see how increasing the K value changes the accuracy.

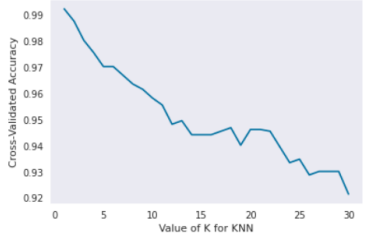
→ **KNN CLASSIFICATION FOR K=3**

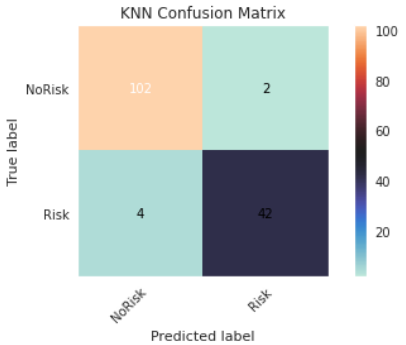
```
[63] neigh = KNeighborsClassifier(n_neighbors=3)
     neigh.fit(x_train, y_train)
     # predict the response
     pred = neigh.predict(x_test)
     # evaluate accuracy
     knnaccuracy = accuracy_score(y_test, pred, normalize = True)
     print ("KNeighbors K=3 classification accuracy score : %0.2f " % knnaccuracy)

     KNeighbors K=3 classification accuracy score : 0.95
```

In the case of K = 3, the models' accuracy decreases to 95%. The classification report and confusion matrix are shown below, to get a better visual idea of how the results are spread out.




```
[60] plt.plot(k_range,k_scores)
     plt.xlabel('Value of K for KNN')
     plt.ylabel('Cross-Validated Accuracy')

     Text(0, 0.5, 'Cross-Validated Accuracy')
```





→ With K=3 for the model, the difference is an additional False Negative correct result
→ Furtherly, the Cross-validation curve of KNN accuracy is plotted.

→ The graph shows that for this dataset, as the K value is increased, there is a decrease in accuracy measurement, dropping to as low as 92% when K is at the maximum tested value of 31.

→ The optimal K value appears to be 1, but that is not a good choice. If K = 1, there will be a higher bias. The training error (bias) will rise as K increases, while the testing error may decrease around the same rate (decrease variance). When it has to include more neighbours as K gets larger, its model gets more complicated. For the reason of keeping the K value as an odd number and avoiding 1.

5.2. DECISION TREE CLASSIFIER

→ The Decision Tree Algorithm (Supervised Learning Algorithm) can be used to solve regression and classification issues. This algorithm provides a decision support tool that employs a tree-like paradigm. The purpose of employing a Decision Tree is to develop a training model that can use that can use basic decision rules derived from training data to predict the target variance.

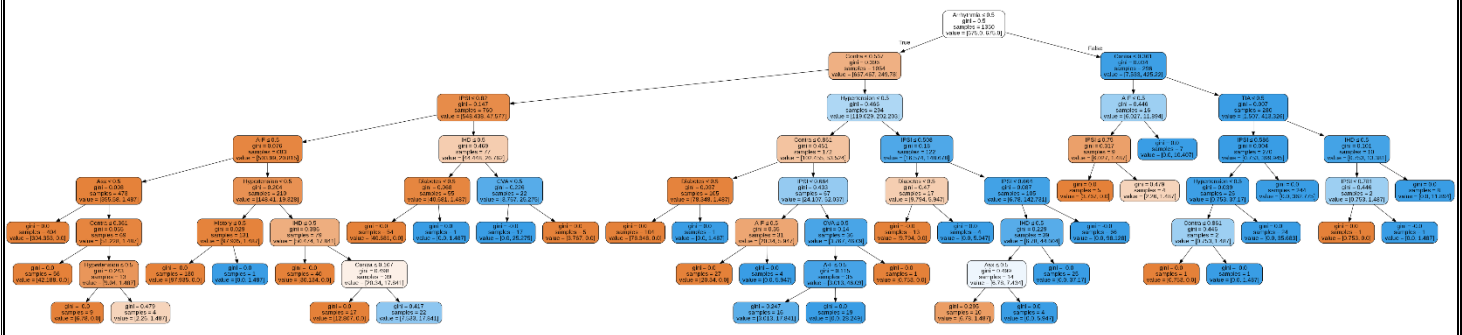
```
[107] #Create the decision tree classifier
      clf = tree.DecisionTreeClassifier(max_depth=7, class_weight='balanced')
      #adjusting max depth increases accuracy, all the way up to depth of 10 = 99% accuracy
      #but this is hard to visually understand

      #Train the classifier
      clf = clf.fit(x_train, y_train)

      #Predict what the response will be for the test dataset
      pred = clf.predict(x_test)

      #Model Accuracy
      treeaccuracy = accuracy_score(y_test, pred, normalize = True)
      print ("Decision Tree classification accuracy score : %0.2f " % treeaccuracy)

      Decision Tree classification accuracy score : 0.98
```

Accuracy is good at 97% with class weight set to balanced. Increasing the max depth will of course increase the accuracy but can introduce overfitting of the model.
The max depth defines how far in the depth the decision tree will go, from top to bottom. As expected, a higher depth will allow a higher prediction accuracy but also results in a higher chance of overfitting skewing the result of the model.

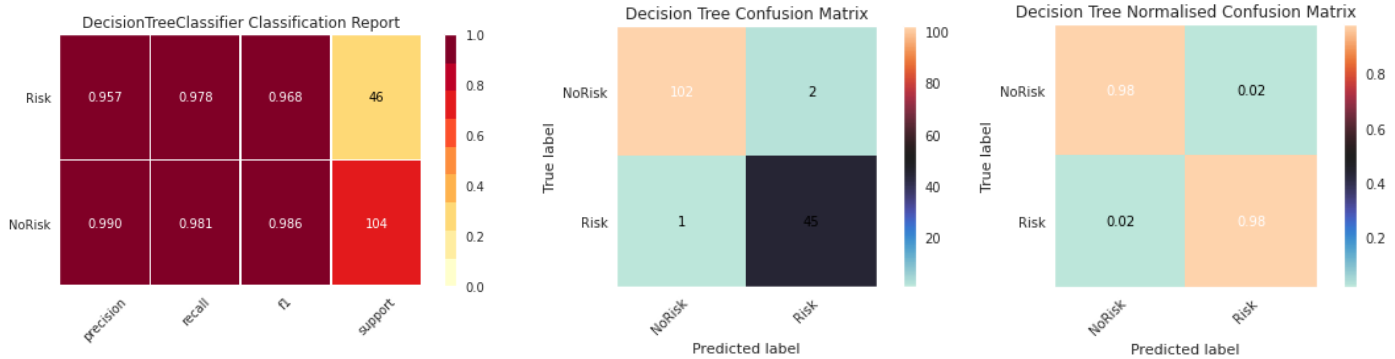
→ To visualize the decision tree, the feature of sklearn. the tree called export_graphviz was exported, which allows us to visualize the graph, and then save that as a file if we want it later on outside of the notebook.

→ The feature names are defined, and the training model is attached to this. This will give a graphical overview of the decision tree and how it comes to its decisions. The default separator for impurity score is Gini, which is not changed for the below visualization. This can be changed to Entropy if desired.

→ The Gini score is a statistical metric that measures how pure a node/leaf is. A Gini score larger than zero indicates that the samples in that node belong to distinct classes.



→ From this decision tree, it's clear how the model trains itself on the data and target given to it. Decision tree classifier starts the initial split at the top of the tree and employs **Arrhythmia** to split at the first step, then uses **Contra**, and so on.



```
clf = tree.DecisionTreeClassifier(max_depth=7, class_weight='balanced')
dtscore = cross_val_score(clf, x, y, cv=50, scoring='accuracy')
print(dtscore)
print("\n")
print("Decision Tree Cross-Validation Accuracy : %0.2f (+/- %0.2f)" % \
      (dtscore.mean(), dtscore.std() * 2))
```

```
[0.97 0.97 1.   1.   1.   0.97 0.9  1.   0.9  1.   1.   0.97 1.   1.
 1.   0.97 1.   0.97 0.93 0.97 1.   0.97 1.   1.   0.97 1.   1.   1.
 0.97 0.97 0.97 1.   0.97 1.   0.9  1.   1.   1.   0.9  0.97 1.   1.
 1.   1.   1.   1.   0.97 1.   0.93 1.  ]


Decision Tree Cross-Validation Accuracy : 0.98 (+/- 0.06)
```

The Decision Tree model does a good job of making the predictions. It is not possible to visualize the Decision Tree for cross-validation without fitting it to the tree, so the accuracy result is looked at instead rather than a visualized tree-like before. Cross-Validation achieves a mean accuracy of 98% with a variance of 0.06. The scores themselves stay near 100% except for a couple of scores at 93% and 97%, which decreases the mean to 98%. Cross-validation is run using the same depth as before of 7. It is possible to visualize the predictions of the Decision Tree in a format such as that of a prediction to an actual data frame, DecisionTreeRegressor function of sklearn can be used for this. Due to the sheer ability to visually examine how the model separates the input to train itself, a decision tree was used. This is significant since it allows one to understand what the model thinks relevant for splitting and supports in determining what is deemed the most critical characteristics. It's clear from the figure using the Decision Tree Classifier we can achieve remarkable accuracy with the most precise result.

```
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)

y_pred = regressor.predict(x_test)

df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df.head(10)
```

	Actual	Predicted
1369	0	0.0
1370	0	0.0
1371	1	1.0
1372	0	0.0
1373	1	1.0
1374	0	0.0
1375	0	0.0
1376	1	1.0
1377	0	0.0
1378	1	1.0

5.3. Neural Network

A Neural Network is essentially a series of algorithms that endeavours to recognise underlying relationship in a set of data through a process that mimics the way human brain operates. Neural Network is a technique popularly used in field of Deep Learning.

For the model deployment, it's started by using a **SDG solver** using the settings below and the learning rate set as **invscaling**. The method '**invscaling'** decreases the learning rate at each step 't' using an inverse scaling exponent of **'power_t'.**

$$Effective\ Learning\ Rate = \frac{Learning\ Rate\_init}{pow(t, power_t)}$$

The **invscaling** didn't work as it shows a very bad effect, in which the model achieves accuracy of about 55% which is less than NULL accuracy (69%). So, for further modelling learning rate was set as adaptive which results into giving a promising accuracy of about 93%. As, we're using Neural Network parameter tunings can be done for increasing the accuracy.

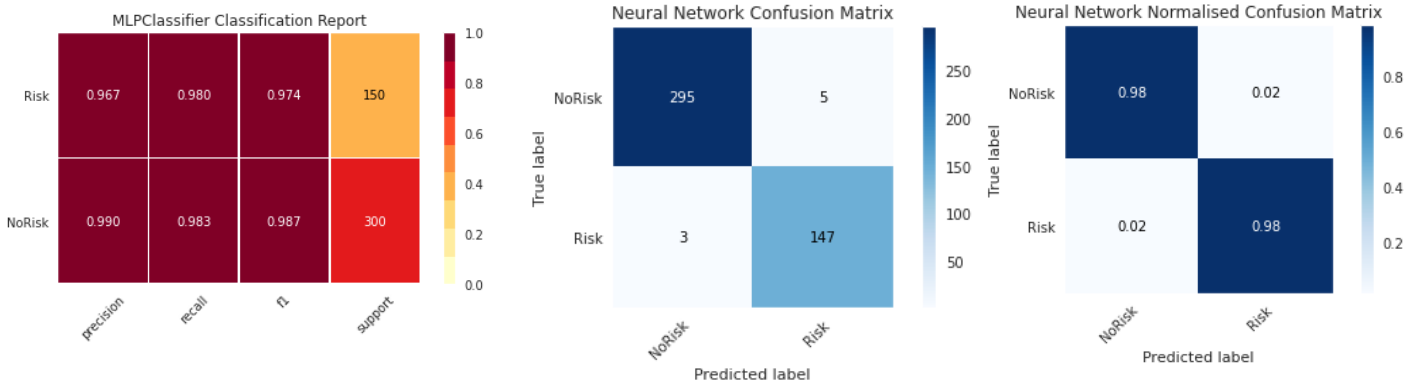
5.3.1. Hyperparameter tuning

⇨ Different hyper parameters were experimented with different values for checking the highest promising accuracy the model can achieve. The final range of values taken are shown in Table x.

Sr. No.	Hyperparameter Configurations	Value
1	Solver	Lgfgs
2	Maximum Iteration	5000
3	Hidden Layer Size	100
4	Batch Size	10
5	Random States	1
6	Learning Rate	Adaptive

With these hyperparameter settings, the model achieved an accuracy of 98% and a training Mean Squared Error score of 0.01777.

Furtherly, the Classification Report and confusion matrix are shown below, to get a better visual idea of how the results are spread out.



⇨ Cross Validation Neural Network score is 99%, with some k-folds hitting 100%. The results are fantastic and very accurate, looking at the overall dataset in a confusion matrix shows that the Neural Network ran against the entire dataset gets a total of 17 predictions wrong out of a total of 1500.

6. COMPARISON ANALYSIS

6.1. RESULT COMPARISON

→ The most essential indicator for this research is Sensitivity, which measures how effectively the model predicts individuals at risk. All the models trained provides a comparable accuracy in the region of 96-97 percent, it's where the Sensitivity score becomes a more essential parameter for distinguishing these algorithms. The sensitivity score or precision measure score are utilized to indicates how effectively a model predicts the percentage of at-risk individuals. The majority of the models have high Sensitivity scores, and it goes without saying that the highest performing model even has the highest Precision values. This statistic is more useful when the models have the same accuracy, which seems to be the case with several of the other models.

K NEAREST NEIGHBOURS	ACCURACY	PREDICTED	
		POSITIVE	**NEGATIVE**
OBSERVED	**POSITIVE**	**101**	**3**
	NEGATIVE	**4**	**42**

K NEAREST NEIGHBOURS CLASSIFICATION REPORT	PRECISION SENSITIVITY	RECALL SPECIFICITY	F1 SCORE
RISK	**93**	**91**	**92**
NORISK	**96**	**97**	**96**

DECISION TREE CLASSIFIER	ACCURACY	PREDICTED	
		POSITIVE	**NEGATIVE**
OBSERVED	**POSITIVE**	**102**	**2**
	NEGATIVE	**1**	**45**

DECISION TREE CLASSIFICATION REPORT	PRECISION SENSITIVITY	RECALL SPECIFICITY	F1 SCORE
RISK	**95**	**97**	**96**
NORISK	**99**	**98**	**98**

7. CONCLUSION AND FUTURE WORK

The purpose of current research is to assess the functionality of machine learning algorithms for estimating the risk of Cardio-vascular disease utilizing data characteristics. We discovered that the Decision Tree is a better algorithm for

predicting risk. Following is a list of qualities that were chosen using a backward elimination procedure that considered P values that were less in percent. The fundamental goal of this study is to compare Classification algorithms to predict patient Risk/No Risk with high accuracy. In conclusion, we can say K Nearest Neighbours is the most accurate model to provide results. Decision Tree ends second, but we preferred KNN over Decision Tree because KNN doesn't take into consideration, the factors that don't play a major role in cardiovascular disease prediction. Decision Tree also takes into consideration, factors such as and, that doesn't play a major role in diabetes prediction. KNN is also better than Decision Trees due to its consistency. Decision Trees under both the indices, do provide us with 99% accuracy but that completely depends on the training data given to it as input. KNN is independent of the training data used and hence provides us with high accuracy under any given Training Data Conditions.

The majority of data is now digital, and it is widely disseminated, yet we are not adequately utilizing it. Analysing the given data may also be used to explore for unknown patterns. The future goal is to utilize Machine Learning Classification Algorithms to predict cardiac illnesses with high accuracy. In machine learning algorithms, we can use Logistic Regression, Support Vector Machine, KNN, Decision Tree, naive Bayes, and sklearn to predict cardiac disease using various factors. Furthermore, data and methodologies could be used to improve the model. The research's future scope is to anticipate cardiac problems utilizing sophisticated methodologies that have a high rate of accuracy, algorithms and are less time-consuming.