A Project Report
on

SURVIVAL OF HEART FAILURE PREDICTION USING FEATURE SCALING

submitted in partial fulfillment of the requirements for the award of the degree
of
BACHELOR OF TECHNOLOGY
in
COMPUTER SICENCE AND ENGINEERING

by

17WH1A0521	**Ms. DANNAPANENI TRIPURA**
17WH1A0556	**Ms. ALLE SHAMINI**
17WH1A0503	**Ms. POLKAM NITHYA SRI**

under the esteemed guidance of
Ms. A. NAGA KALYANI
Assistant Professor

Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

May, 2021

DECLARATION

We hereby declare that the work presented in this project entitled **"SURVIVAL OF HEART FAILURE PREDICTION USING FEATURE SCALING"** submitted towards completion of Project Work in IV year of B.Tech., CSE at 'BVRIT HYDERABAD College of Engineering for Women', Hyderabad is an authentic record of our original work carried out under the guidance of Ms. Naga Kalyani, Assistant Professor, Department of CSE.

Sign. with date:
Ms. DANNAPANENI TRIPURA
(17WH1A0521)

Sign. with date:
Ms. ALLE SHAMINI
(17WH1A0556)

Sign. with date:
Ms. POLKAM NITHYA SRI
(17WH1A0503)

BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering

Certificate

This is to certify that the Project Work report on "**SURVIVAL OF HEART FAILURE PREDICTION USING FEATURE SCALING**" is a bonafide work carried out by Ms. DANNAPANENI TRIPURA (17WH1A0521) ; Ms. ALLE SHAMINI (17WH1A0556) ; Ms. POLKAM NITHYA SRI (17WH1A0503) in the partial fulfillment for the award of B.Tech. degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department
Dr. Srinivasa Reddy Konda
Professor and HoD,
Department of CSE

Guide
Ms. A. Naga Kalyani
Assistant Professor

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal**, **BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. Srinivasa Reddy Konda, Professor**, Department of CSE, **BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Ms. A. Naga Kalyani, Assistant Professor**, Department of CSE**, BVRIT HYDERABAD College of Engineering for Women** for her constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of **CSE** Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Ms. DANNAPANENI TRIPURA
(17WH1A0521)

Ms. ALLE SHAMINI
(17WH1A0556)

Ms. POLKAM NITHYA SRI
(17WH1A0503)

CONTENTS

ABSTRACT

Cardiovascular disease is a class of diseases that involve the heart or blood vessels. One such cardiovascular disease is heart failure. Heart failure is a condition when the heart muscle doesn't pump blood as well as it should. It is a major public health problem in adult population in developed countries.

Machine learning techniques when applied to the medical records can predict patient's survival and rank the features corresponding to the most important risk factors analysis. A feature ranking analysis which includes serum creatinine and ejection fraction can be employed to traditional biostatistics tests, and compare these results with those provided by the machine learning algorithms.

LIST OF FIGURES

1. INTRODUCTION

Heart failure prediction deals with predicting the danger level of the patient having heart failure. It can be used as an effective tool, both to predict the survival of each patient having heart failure symptoms and to detect the most important clinical features (or risk factors) that may lead to heart failure by using machine learning. Machine learning is a branch of artificial intelligence (AI) focused on building applications that learn from data.

1.1 Objectives

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide. Given the importance of a vital organ such as the heart, predicting heart failure has become a priority for medical doctors and physicians. Electronic health records (EHRs, also called medical records) can be considered as a useful resource of information to unveil hidden and non-obvious correlations and relationships between patients' data, not only for research but also for clinical practice and for debunking traditional myths on risk factors. In this project, prediction of survival rate helps doctors to get an estimate with regards to deterioration or improvement in the patient's condition. This in turn allows them to make decisions with respect to future course of treatment that would be required.

1.2 Methodology

To predict the survival rate of a patient, a collection of patients health details is required. The dataset is downloaded from Kaggle. In this section the methodology followed is discussed in detail.

1.2.1 Dataset

Dataset consists of 13 features and 299 instances of heart failure patients collected at the Faisalabad Institute of Cardiology and at the Allied Hospital in Faisalabad (Punjab, Pakistan), during April–December 2015.

The patients consist of 105 women and 194 men and their ranges range between 40 and 95 years old. All 299 patients had previous heart failures. Some features like anemia, high blood pressure, diabetes, sex and smoking are binary.

Feature	Explanation
Age	Age of patient
Anaemia	Decrease of red blood cells
High blood pressure	If a patient has hypertension
Creatinine phosphokinase	Level of CPK enzyme in the blood
Diabetes	If a patient has diabetes
Ejection fraction	Percentage of blood leaving the heart at each contraction
Sex	Woman or man
Platelets	Platelets in the blood
Serum creatinine	Level of creatinine in the blood
Serum sodium	Level of sodium in the blood
Smoking	If the patient smokes
Time	Follow-up period
Death Event	If the patient died during the follow-up period

Fig 1.2.1.1 Dataset details

1.2.2 The Proposed models

Random Forest

Random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting.

It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

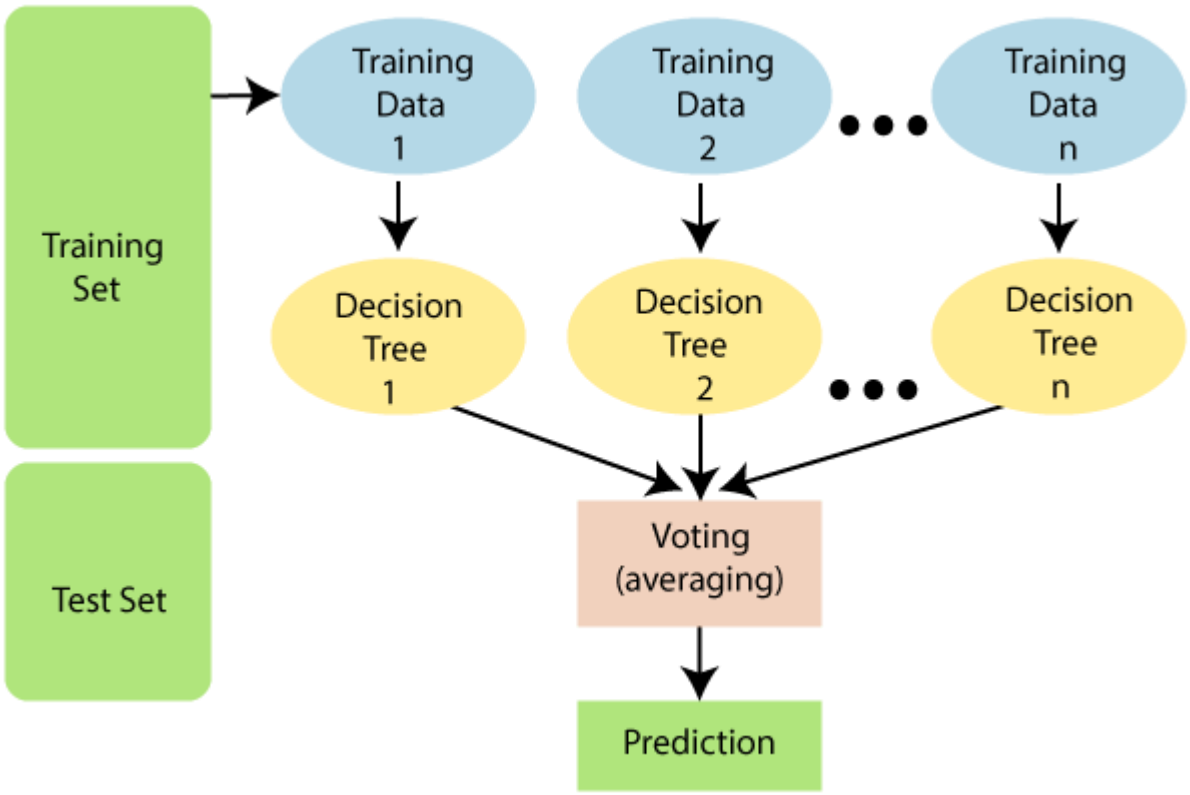

Fig 1.2.2.1 Working of Random Forest Algorithm

Advantages:

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

Disadvantages:

- Random forest algorithm may change considerably by a small change in the data.

Support Vector Machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

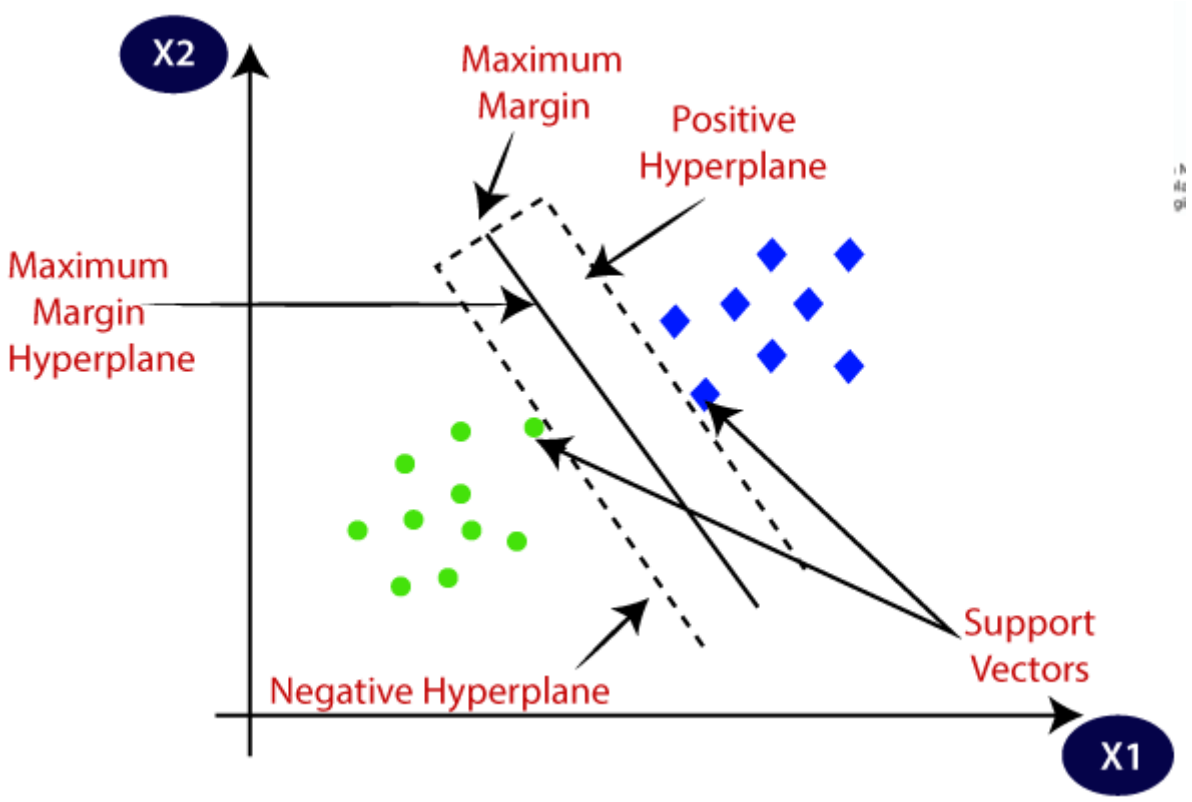

Fig 1.2.2.2 Support Vector Machine

SVM can be of two types**:**

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Advantages:

- SVM works relatively well when there is a clear margin of separation between classes.
- SVM is more effective in high dimensional spaces.
- SVM is relatively memory efficient

Disadvantages:

- SVM does not perform very well when the data set has more noise i.e. target classes are overlapping.
- In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.
- As the support vector classifier works by putting data points, above and below the classifying hyperplane there is no probabilistic explanation for the classification.

Logistic Regression

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes.

In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).

Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

Logistic Function (Sigmoid function)

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

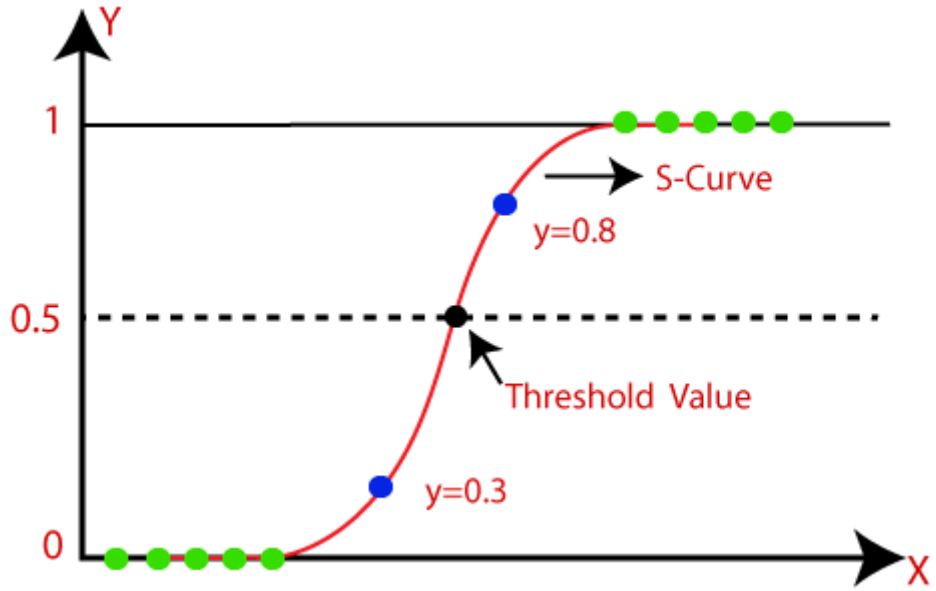

Fig 1.2.2.3 Graph of Logistic Function

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}\text{; 0 for y= 0, and infinity for y=1}$$

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

Logistic regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Advantages:

- Logistic regression is easier to implement, interpret, and very efficient to train.
- It makes no assumptions about distributions of classes in feature space.
- It can easily extend to multiple classes (multinomial regression) and a natural probabilistic view of class predictions.

Disadvantages:

- The major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables.
- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise, it may lead to overfitting.

Decision Tree Algorithm

Decision tree algorithm is a supervised learning algorithm, which can be used for solving both regression and classification problems. The goal of using decision tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules. For predicting a class label for a record, we start from the root of the tree. This algorithm compares the values of the root attribute with the records attribute.

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

- **Branch/Sub Tree:** A tree formed by splitting the tree.

- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.

- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

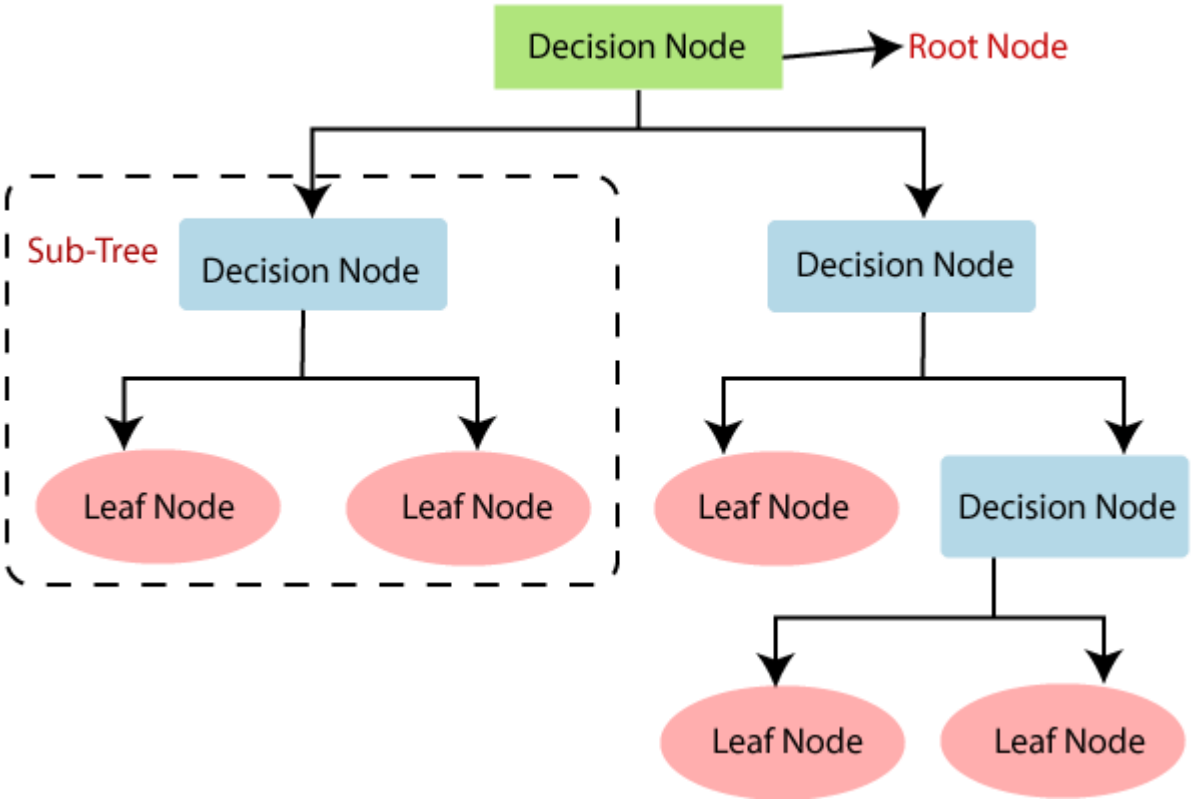

Fig 1.2.2.4 General Structure of Decision Tree

Advantages:

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages:

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the Random Forest algorithm.
- For more class labels, the computational complexity of the decision tree may increase.

K-Nearest Neighbour

K-nearest neighbours (KNN) algorithm is a type of supervised Machine Learning algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties will define KNN −

- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

KNN algorithm assumes that similar things exist in close proximity. In other words similar things are near to each other.

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset.

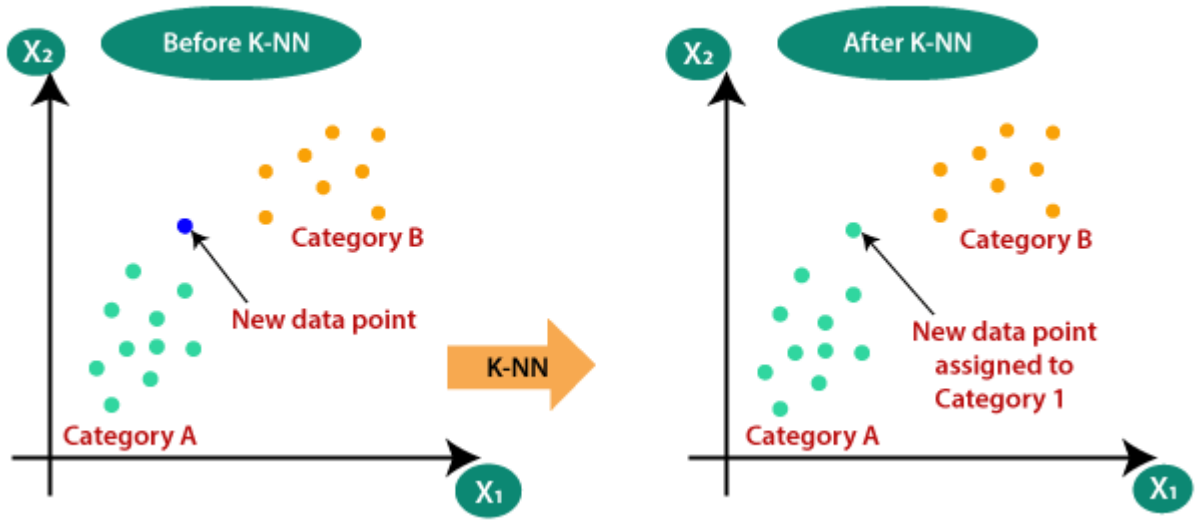

Fig 1.2.2.5 K-nearest neighbour

Advantages:

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

Disadvantages:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

1.3 Organization of Project

The technique which is developed here is, taking input values for serum creatinine and ejection fraction, based on these values the trained models will predict the death event and then finally display the survival rate of that patient by performing average of predicted death events.

2. THEORITICAL ANALYSIS OF PROPOSED PROJECT

2.1 Requirements Gathering

2.1.1 Software Requirements

Programming Language : Python

Tool : Google Colaboratory

Framework : Pycharm

Packages : Pandas, Scikit-Learn, Scipy, Prettytable

2.1.2 Hardware Requirements

Operating System : Windows 10

Processor : Intel Core i5

Memory : 1 Tb

2.2 Technologies Description

Python

Python is an interpreted high-level general-purpose programming language. It is used in web development, data science, creating software prototypes etc. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. It works on different platforms, runs on an interpreter system i.e. code can be executed as soon as it is written due to this prototyping can be very quick. Its large standard library, provides tools suitable to many tasks.

Pandas

Pandas is an open-source, Python library providing high-performance, easy-to-use data structures and data analysis tools for python programming language. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyse. There are many features of pandas like tools for loading data into in-memory data

objects from different file formats, reshaping and pivoting data sets, columns from data structure can be deleted or inserted.

Scikit-learn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. Rather than focusing on loading, manipulating and summarising data, Scikit-learn library is focused on modelling the data. Some of the most popular groups of models provided by Sklearn are as follows – Supervised learning algorithms, unsupervised learning algorithms, feature selection, clustering etc.

Scipy

SciPy, a scientific library for Python is an open source. It is used to solve the complex scientific and mathematical problems. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The main reason for building the SciPy library is that, it should work with NumPy arrays. It provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization.

Prettytable

PrettyTable is a Python library for generating simple ASCII tables. It can control many aspects of a table, such as the width of the column padding, the alignment of text, or the table border. It can sort data. It can also choose which columns and rows are going to be displayed in the final output. PrettyTable can read data from CSV, HTML, or database cursor and output data in ASCII or HTML.

Google Colaboratory

Google Colaboratory allows you to write and execute Python in your browser, with zero configuration required, free access to GPUs, easy sharing. Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them.

Pycharm

PyCharm is the most popular IDE used for Python scripting language. It offers some of the best features to its users and developers in the following aspects code completion and inspection, advanced debugging, support for web programming and frameworks such as Django and Flask. PyCharm is created by Czech company, Jet brains which focusses on creating integrated development environment for various web development languages like JavaScript and PHP.

Matplotlib

Matplotlib is an open source Python package used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. It provides an object-oriented API that helps in embedding plots in applications. It can be used in Python and IPython shells, Jupyter notebook and web application servers also.

Tkinter

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps − Import the Tkinter module. Create the GUI application main window. Add one or more of the above-mentioned widgets to the GUI application. Enter the main event loop to take action against each event triggered by the user.

3. DESIGN

3.1 Architecture Diagram

Considered a dataset of heart failure patients. Then analysed the dataset. The dataset we considered doesn't have any missing values, but each feature is in different scale so performed normalization on all the features and then, performed feature scaling by using Heatmap, Pearson correlation coefficient, Chi square values to get the features which have high dependency on target value. Then implemented algorithms on training data like Random Forest, Support Vector Machine (SVM), Logistic Regression, Decision Tree, K-nearest Neighbours to train the model and calculated accuracy and mean square error of each model. By using these models, developed a UI page which takes input from users and display the danger level.

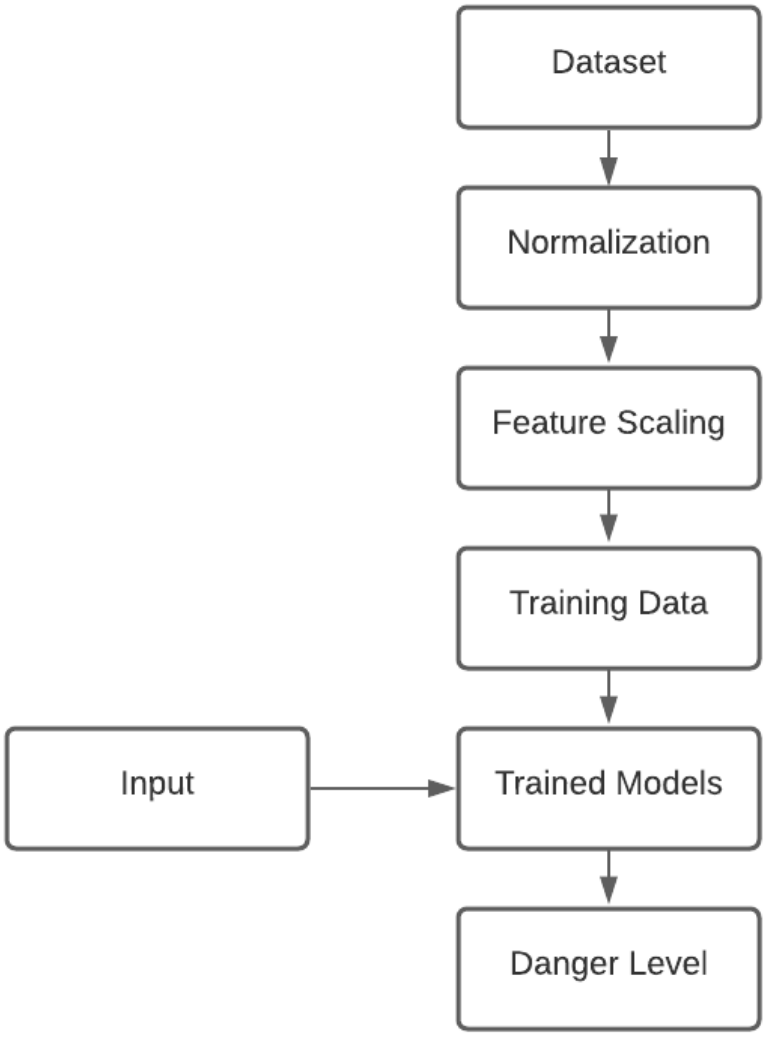

Fig 3.1 Architecture Diagram

3.2 UML Diagrams

3.2.1 Use Case Diagram

A use case diagram is a behaviour diagram and visualizes the observable interactions between actors and the system under development. The diagram consists of the system, related use cases and actors and relates these to each other. It does not show the order in which steps are performed to achieve the goals of each use case.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analysed to gather its functionalities, use cases are prepared and actors are identified.

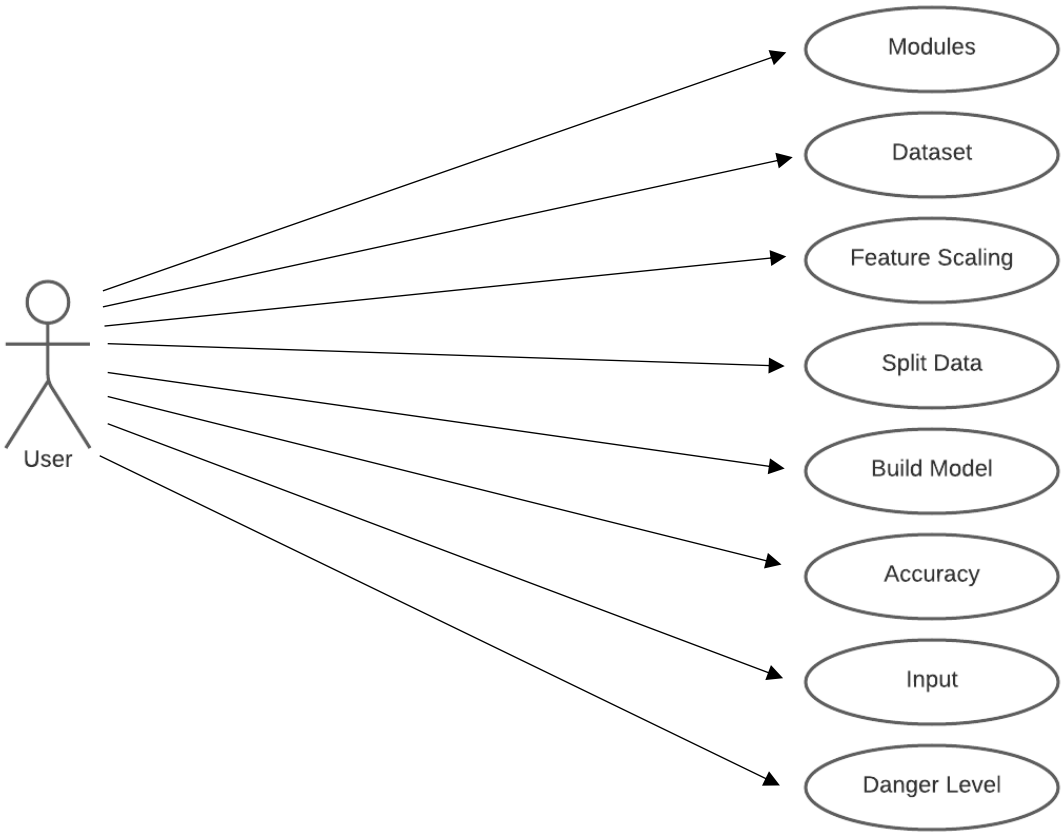

Fig 3.2.1 Use case diagram

3.2.2 Sequence Diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They are also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios. These can help to predict how a system will behave and to discover responsibilities a class may need to have in the process of modelling a new system. In sequence diagram, lifeline is represented by a vertical bar, message flow is represented by vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

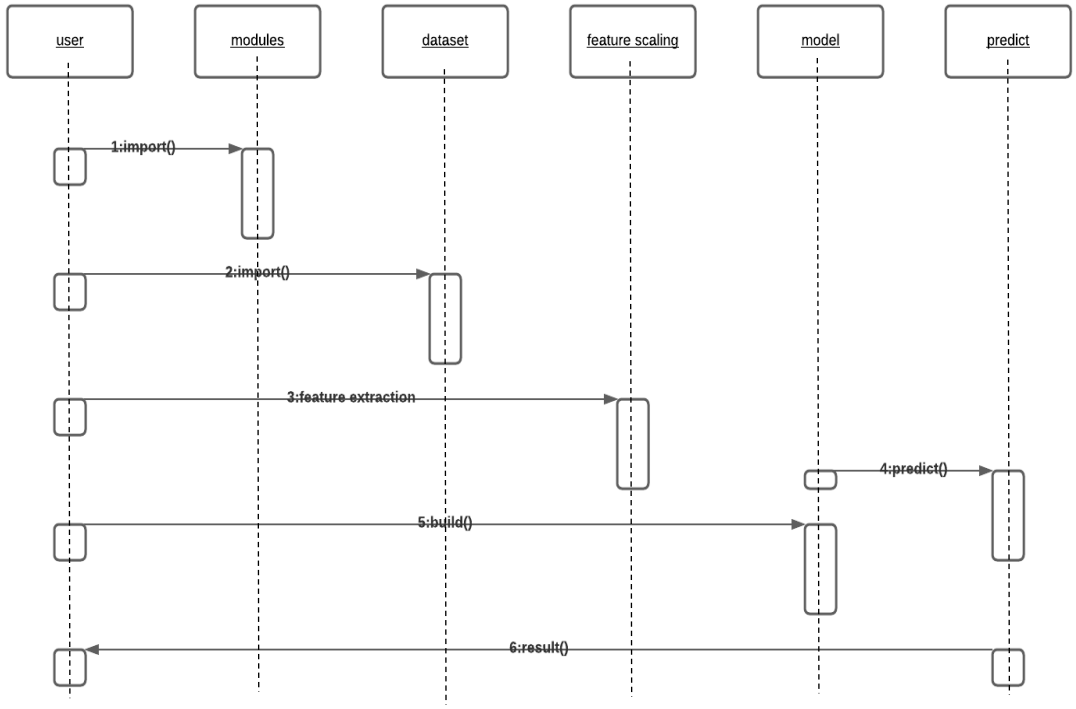

Fig 3.2.2 Sequence Diagram

3.2.3 Activity Diagram

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagram deals with all type of flow control.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.

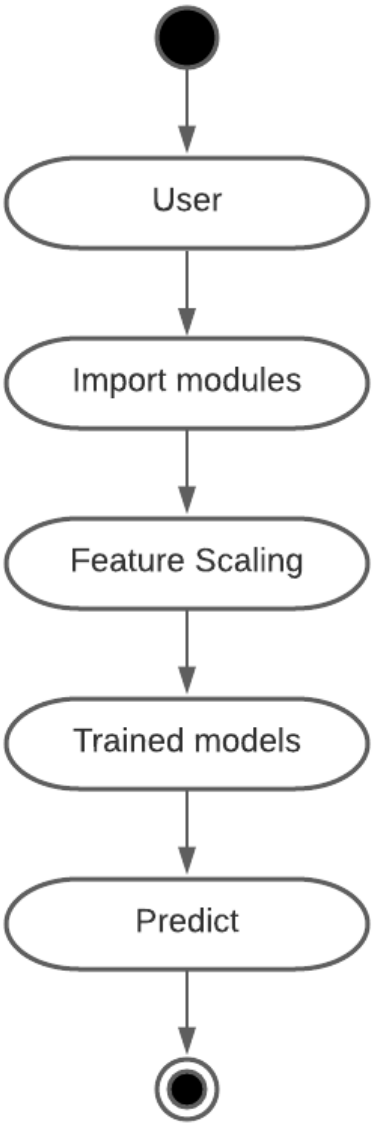

Fig 3.2.3 Activity Diagram

3.2.4 Class Diagram

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consist of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

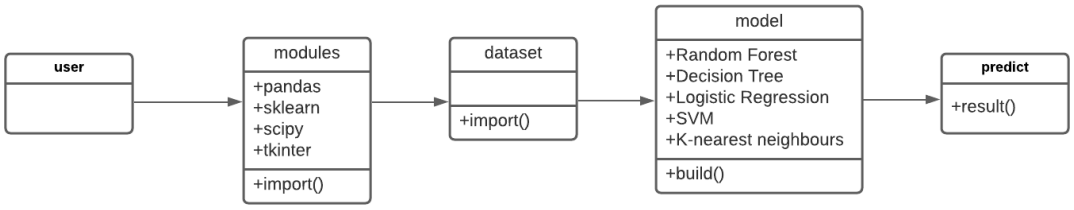

Fig 3.2.4 Class Diagram

4. IMPLEMENTATION

4.1 Coding

Survival of Heart Failure Prediction using Feature Scaling.ipynb

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from prettytable import PrettyTable

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

import operator
from scipy.stats import pearsonr
from scipy.stats import chi2_contingency

from google.colab import drive
drive.mount('/content/drive')
```

```
data = pd.read_csv('drive/My Drive/MajorProject/S1Data.csv')
```

```
data.head()
```

	TIME	Event	Gender	Smoking	Diabetes	BP	Anaemia	Age	Ejection.Fraction	Sodium	Creatinine	Pletelets	CPK
0	97	0	0	0	0	0	1	43.0	50	135	1.30	237000.00	358
1	180	0	1	1	1	0	1	73.0	30	142	1.18	160000.00	231
2	31	1	1	1	0	1	0	70.0	20	134	1.83	263358.03	582
3	87	0	1	0	0	0	1	65.0	25	141	1.10	298000.00	305
4	113	0	1	0	0	0	0	64.0	60	137	1.00	242000.00	1610

Fig 4.1.1 Dataset

```
data.shape
```

```
(299, 13)
```

Fig 4.1.2 Shape of Dataset

```
data.isnull().sum()
```

```
TIME                0
Event               0
Gender              0
Smoking             0
Diabetes            0
BP                  0
Anaemia             0
Age                 0
Ejection.Fraction   0
Sodium              0
Creatinine          0
Pletelets           0
CPK                 0
dtype: int64
```

Fig 4.1.3 Missing values in dataset

```
print("Minimum value of each feature\n", data.min(), "\nMaximum value of each feature\n", data.max())
```

```
Minimum value of each feature
 TIME                      4.0
Event                      0.0
Gender                     0.0
Smoking                    0.0
Diabetes                   0.0
BP                         0.0
Anaemia                    0.0
Age                       40.0
Ejection.Fraction         14.0
Sodium                   113.0
Creatinine                 0.5
Pletelets              25100.0
CPK                       23.0
dtype: float64
Maximum value of each feature
 TIME                     285.0
Event                      1.0
Gender                     1.0
Smoking                    1.0
Diabetes                   1.0
BP                         1.0
Anaemia                    1.0
Age                       95.0
Ejection.Fraction         80.0
Sodium                   148.0
Creatinine                 9.4
Pletelets             850000.0
CPK                    7861.0
dtype: float64
```

Fig 4.1.4 Range of each feature in dataset

```
X=data.drop(['Event'],axis=1)

y=data['Event']

x = preprocessing.normalize(X)

X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.3,random_state=0)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)


table = PrettyTable()

normalization_table = PrettyTable()

table.field_names = ["Model", "Mean Squared Error", "Test Score", "Train Score"]

normalization_table.field_names = ["Model", "Mean Squared Error", "Test Score", "Train Sc
ore"]
```

```
models = [
    RandomForestClassifier(n_estimators=17,max_depth=3),
    SVC(kernel='linear') ,
    LogisticRegression(),
    DecisionTreeClassifier(),
    KNeighborsClassifier()
]
for model in models:
    model.fit(x_train, y_train)
    y_res = model.predict(x_test)
    mse_n = mean_squared_error(y_test, y_res)
    test_score_n = model.score(x_test, y_test)
    train_score_n = model.score(x_train, y_train)
    normalization_table.add_row([type(model).__name__, format(mse_n, '.2f'), format(test_sc
ore_n, '.2f'), format(train_score_n, '.2f')])
    model.fit(X_train, Y_train)
    Y_res = model.predict(X_test)
    mse = mean_squared_error(Y_test, Y_res)
    test_score = model.score(X_test, Y_test)
    train_score = model.score(X_train, Y_train)
    table.add_row([type(model).__name__, format(mse, '.2f'), format(test_score, '.2f'), format(t
rain_score, '.2f')])


print("Accuracy values before normalization\n", table, "\nAccuracy values after normalizatio
n\n", normalization_table)
```

Accuracy values before normalization

Model	Mean Squared Error	Test Score	Train Score
RandomForestClassifier	0.14	0.86	0.90
SVC	0.20	0.80	0.79
LogisticRegression	0.12	0.88	0.82
DecisionTreeClassifier	0.21	0.79	1.00
KNeighborsClassifier	0.43	0.57	0.74

Accuracy values after normalization

Model	Mean Squared Error	Test Score	Train Score
RandomForestClassifier	0.21	0.79	0.85
SVC	0.36	0.64	0.69
LogisticRegression	0.36	0.64	0.69
DecisionTreeClassifier	0.19	0.81	1.00
KNeighborsClassifier	0.32	0.68	0.82

Fig 4.1.5 Accuracy and mean square error after implementing algorithms on all features

```
corr_matrix = data.corr()

top_corr_feature = corr_matrix.index

plt.figure(figsize=(15, 15))

sns.heatmap(data[top_corr_feature].corr(), annot=True, cmap="RdYlGn", annot_kws={"size
":15})

data.corr()['Event'].sort_values(ascending=False)
```

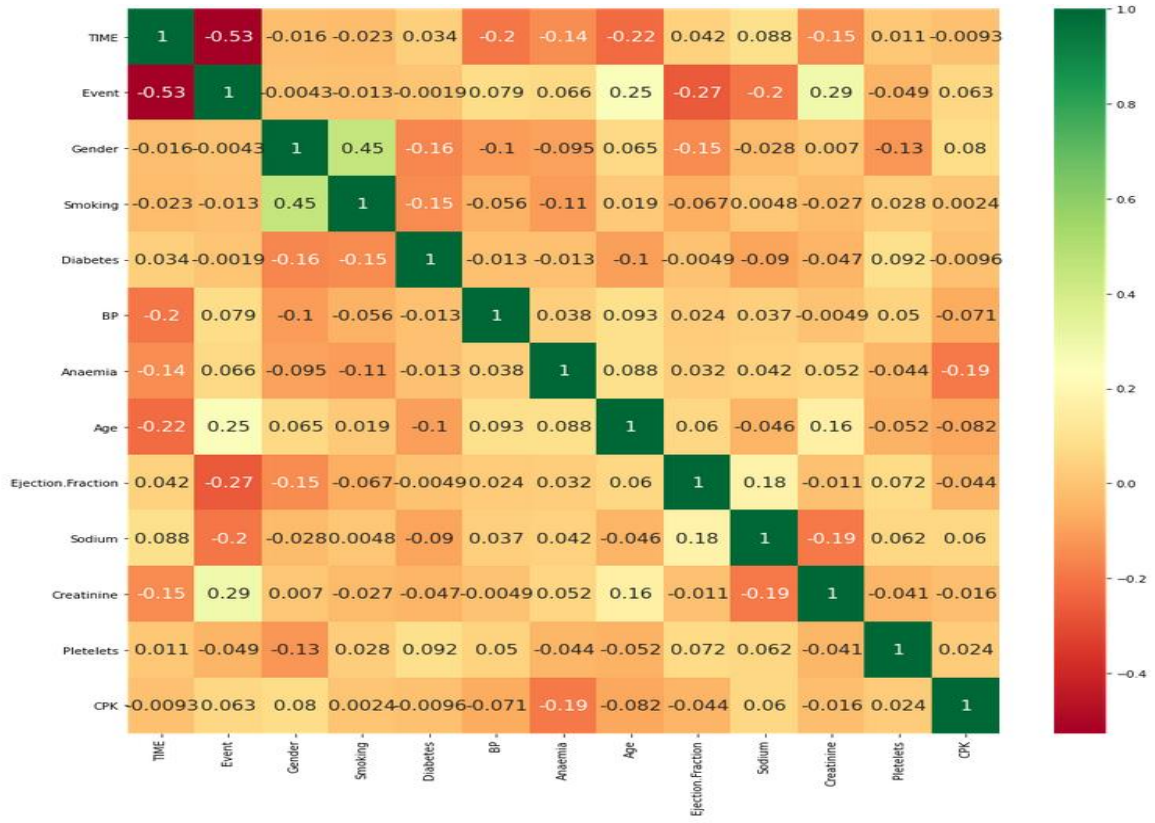

Fig 4.1.6 Heatmap

```
d = {}
for i in data:
  corr, _ = pearsonr(data[i], data["Event"])
  d[i] = abs(corr)
sorted_d = dict( sorted(d.items(), key=operator.itemgetter(1),reverse=True))
for i in sorted_d:
  print(i, sorted_d[i])
```

```
Event 0.9999999999999999
TIME 0.5269637792775768
Creatinine 0.29427756098414926
Ejection.Fraction 0.26860331239406204
Age 0.25372854308800363
Sodium 0.1952035964164012
BP 0.07935105769128536
Anaemia 0.06627009846028778
CPK 0.06272816025237477
Pletelets 0.04913886798037428
Smoking 0.012623152709359632
Gender 0.004316376319703108
Diabetes 0.0019428833442034 77
```

Fig 4.1.7 Pearson correlation coefficient values

```
di = {}
for i in data:
  d = pd.crosstab(data[i], data['Event'])
  stat, p, dof, expected = chi2_contingency(d)
  #print(i, str(p))
  di[i] = p
sorted_d = dict( sorted(di.items(), key=operator.itemgetter(1)))

for i in sorted_d:
  print(i, sorted_d[i])
```

```
Event 5.386429328048915e-66
Ejection.Fraction 6.459327810543188e-08
TIME 6.590460363385011e-07
Creatinine 3.1452364264229703e-06
Sodium 0.009600557349298953
Age 0.015227406432460549
BP 0.21410341199416902
Anaemia 0.3073160508415107
CPK 0.43175059718299313
Pletelets 0.5482703513757745
Diabetes 0.9267235137291102
Smoking 0.9317652998235507
Gender 0.9560508538247334
```

Fig 4.1.8 Chi square values

```
X=data.drop(['CPK', 'Pletelets', 'Diabetes', 'Smoking', 'Gender', 'Event'],axis=1)
y=data['Event']
x = preprocessing.normalize(X)
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.3,random_state=0)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)


table = PrettyTable()
normalization_table = PrettyTable()
table.field_names = ["Model", "Mean Squared Error", "Test Score", "Train Score"]
normalization_table.field_names = ["Model", "Mean Squared Error", "Test Score", "Train Sc
ore"]


models = [
    RandomForestClassifier(n_estimators=17,max_depth=3),
    SVC(kernel='linear') ,
    LogisticRegression(),
    DecisionTreeClassifier(),
    KNeighborsClassifier()
]
```

```
for model in models:
    model.fit(x_train, y_train)
    y_res = model.predict(x_test)
    mse_n = mean_squared_error(y_test, y_res)
    test_score_n = model.score(x_test, y_test)
    train_score_n = model.score(x_train, y_train)
    normalization_table.add_row([type(model).__name__, format(mse_n, '.2f'), format(test_score_n, '.2f'), format(train_score_n, '.2f')])
    model.fit(X_train, Y_train)
    Y_res = model.predict(X_test)
    mse = mean_squared_error(Y_test, Y_res)
    test_score = model.score(X_test, Y_test)
    train_score = model.score(X_train, Y_train)
    table.add_row([type(model).__name__, format(mse, '.2f'), format(test_score, '.2f'), format(train_score, '.2f')])
print("Accuracy values before normalization\n", table, "\nAccuracy values after normalization\n", normalization_table)
```

```
Accuracy values before normalization
+------------------------+--------------------+------------+-------------+
|         Model          | Mean Squared Error | Test Score | Train Score |
+------------------------+--------------------+------------+-------------+
| RandomForestClassifier |        0.16        |    0.84    |     0.89    |
|          SVC           |        0.18        |    0.82    |     0.84    |
|   LogisticRegression   |        0.18        |    0.82    |     0.84    |
| DecisionTreeClassifier |        0.21        |    0.79    |     1.00    |
|  KNeighborsClassifier  |        0.11        |    0.89    |     0.86    |
+------------------------+--------------------+------------+-------------+
Accuracy values after normalization
+------------------------+--------------------+------------+-------------+
|         Model          | Mean Squared Error | Test Score | Train Score |
+------------------------+--------------------+------------+-------------+
| RandomForestClassifier |        0.14        |    0.86    |     0.87    |
|          SVC           |        0.13        |    0.87    |     0.83    |
|   LogisticRegression   |        0.16        |    0.84    |     0.83    |
| DecisionTreeClassifier |        0.19        |    0.81    |     1.00    |
|  KNeighborsClassifier  |        0.13        |    0.87    |     0.87    |
+------------------------+--------------------+------------+-------------+
```

Fig 4.1.9 Accuracy and mean square error after feature scaling

```
X=data[['Creatinine', 'Ejection.Fraction']]
x = preprocessing.normalize(X)
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.3,random_state=0)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)

table = PrettyTable()
normalization_table = PrettyTable()
table.field_names = ["Model", "Mean Squared Error", "Test Score", "Train Score"]
normalization_table.field_names = ["Model", "Mean Squared Error", "Test Score", "Train Sc
ore"]

models = [
    RandomForestClassifier(n_estimators=17,max_depth=3),
    SVC(kernel='linear') ,
    LogisticRegression(),
    DecisionTreeClassifier(),
    KNeighborsClassifier()
]

for model in models:
    model.fit(x_train, y_train)
    y_res = model.predict(x_test)
    mse_n = mean_squared_error(y_test, y_res)
    test_score_n = model.score(x_test, y_test)
    train_score_n = model.score(x_train, y_train)
    normalization_table.add_row([type(model).__name__, format(mse_n, '.2f'), format(test_sc
ore_n, '.2f'), format(train_score_n, '.2f')])
    model.fit(X_train, Y_train)
    Y_res = model.predict(X_test)
    mse = mean_squared_error(Y_test, Y_res)
    test_score = model.score(X_test, Y_test)
```

```
    train_score = model.score(X_train, Y_train)
    table.add_row([type(model).__name__, format(mse, '.2f'), format(test_score, '.2f'), format(t
rain_score, '.2f')])
print("Accuracy values before normalization\n", table, "\nAccuracy values after normalizatio
n\n", normalization_table)
```

```
Accuracy values before normalization
 +------------------------+--------------------+------------+-------------+
|         Model          | Mean Squared Error | Test Score | Train Score |
+------------------------+--------------------+------------+-------------+
| RandomForestClassifier |        0.26        |    0.74    |     0.80    |
|          SVC           |        0.31        |    0.69    |     0.71    |
|   LogisticRegression   |        0.30        |    0.70    |     0.77    |
| DecisionTreeClassifier |        0.32        |    0.68    |     0.91    |
|  KNeighborsClassifier  |        0.21        |    0.79    |     0.82    |
+------------------------+--------------------+------------+-------------+
Accuracy values after normalization
 +------------------------+--------------------+------------+-------------+
|         Model          | Mean Squared Error | Test Score | Train Score |
+------------------------+--------------------+------------+-------------+
| RandomForestClassifier |        0.27        |    0.73    |     0.80    |
|          SVC           |        0.36        |    0.64    |     0.69    |
|   LogisticRegression   |        0.36        |    0.64    |     0.69    |
| DecisionTreeClassifier |        0.32        |    0.68    |     0.88    |
|  KNeighborsClassifier  |        0.28        |    0.72    |     0.78    |
+------------------------+--------------------+------------+-------------+
```

Fig 4.1.10 Accuracy and mean square error after implementing algorithms on serum creatinine and ejection fraction

Heart Failure Prediction.py

```
from tkinter import *
from tkinter import messagebox
import pandas as pd
from sklearn.model_selection import train_test_split

from prettytable import PrettyTable
from sklearn import preprocessing
from sklearn.svm import SVC
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

window = Tk()
window.title("Survival of Heart Failure Prediction using Feature Scaling")

options = [
   "Yes",
   "No",
]

label1 = Label(window, text = "Fill the form")
label1.place(x = 500, y = 20)

label2 = Label(window, text = "Enter the creatiene value")
label2.place(x = 550, y = 100)
creatiene = Entry(window, width = 20)
creatiene.place(x = 720, y = 100)

label3 = Label(window, text = "Enter the ejection value")
label3.place(x = 550, y = 150)
ejection = Entry(window, width = 20)
ejection.place(x = 720, y = 150)

label4 = Label(window, text = "Enter the sodium value")
label4.place(x = 550, y = 200)
sodium = Entry(window, width = 20)
sodium.place(x = 720, y = 200)
```

```
label5 = Label(window, text = "Enter the age of patient")
label5.place(x = 550, y = 250)
age = Entry(window, width = 20)
age.place(x = 720, y = 250)

label6 = Label(window, text = "Does the patient have anemia")
label6.place(x = 550, y = 300)
anemia = StringVar()
anemia.set("Yes")
drop_anemia = OptionMenu(window, anemia, *options)
drop_anemia.place(x = 720, y = 300);

label7 = Label(window, text = "Does the patient have BP")
label7.place(x = 550, y = 350)
BP = StringVar()
BP.set("Yes")
drop_bp = OptionMenu(window, BP, *options)
drop_bp.place(x = 720, y = 350);

label8 = Label(window, text = "Enter the follow-up time(days)")
label8.place(x = 550, y = 400)
time = Entry(window, width = 20)
time.place(x = 720, y = 400)

def validation():
   if(creatiene.get() == ""):
      messagebox.showwarning("Required", "Please fill creatiene value")
   elif(ejection.get() == ""):
      messagebox.showwarning("Required", "Please fill ejection value")
```

```
    elif(sodium.get() == ""):
        messagebox.showwarning("Required", "Please fill sodium value")
    elif(age.get() == ""):
        messagebox.showwarning("Required", "Enter patient age")
    elif(time.get() == ""):
        messagebox.showwarning("Required", "Please fill time")
    else:
        display()


def display():
    data = pd.read_csv('C:/Users/91949/Desktop/MAJOR PROJECT/S1Data.csv')
    creatiene_ = creatiene.get()
    ejection_ = ejection.get()
    sodium_ = sodium.get()
    age_ = age.get()
    anemia_ = anemia.get()
    if(anemia_ == "Yes"):
        anemia_ = 1
    else:
        anemia_ = 0
    bp = BP.get()
    if(bp == "Yes"):
        bp = 1
    else:
        bp = 0
    Time = time.get()
    input_values = [[Time, bp, anemia_, age_, ejection_, sodium_, creatiene_]]
    x = data[['TIME', 'BP', 'Anaemia', 'Age', 'Ejection.Fraction', 'Sodium', 'Creatinine']]
    y = data['Event']
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

```
    table = PrettyTable()
    table.field_names = ["Model", "Death event"]
    models = [
        RandomForestClassifier(n_estimators=17, max_depth=3),
        SVC(kernel='linear'),
        LogisticRegression(),
        DecisionTreeClassifier(),
        KNeighborsClassifier()
    ]
    survival_rate = 0
    for model in models:
        model.fit(x_train, y_train)
        y_res = model.predict(input_values)
        print(y_res[0])
        survival_rate = survival_rate + y_res[0]
        table.add_row([type(model).__name__, format(y_res)])
    danger_level = ["Very High Danger", "High Danger", "Considerable Danger", "Moderate
Danger", "Minor Danger", "Little Danger"]
    messagebox.showinfo('Danger Level', danger_level[survival_rate])
Button(window, text = "Submit", command = validation).place(x = 550, y = 450)
window.mainloop()
```

4.2 Test Cases

Test Case ID	Test Scenario	Expected Result	Actual/Result	Pass/Fail
TCO1	Check whether new tab is displayed after running the code.	Tab should be displayed with form in it.	As Expected	Pass
TCO2	Check whether values are getting submitted after clicking on submit	Another tab should be opened	As Expected	Pass
TCO3	Check whether very high danger level is displaying.	Notification tab should be displayed	As Expected	Pass
TCO4	Check whether very high danger level is displaying	After submitting it should display very high danger	As Expected	Pass
TCO5	Check whether high level is displaying	After submitting it should display high danger	As Expected	Pass
TCO6	Check whether considerable level is displaying	After submitting it should display considerable danger	As Expected	Pass
TCO7	Check whether moderate level is displaying	After submitting it should display moderate danger	As Expected	Pass
TCO8	Check whether minor level is displaying	After submitting it should display minor danger	As Expected	Pass
TCO9	Check whether little level is displaying	After submitting it should display little danger	As Expected	Pass

Fig 4.2.1 Test cases

4.3 Output Screenshots

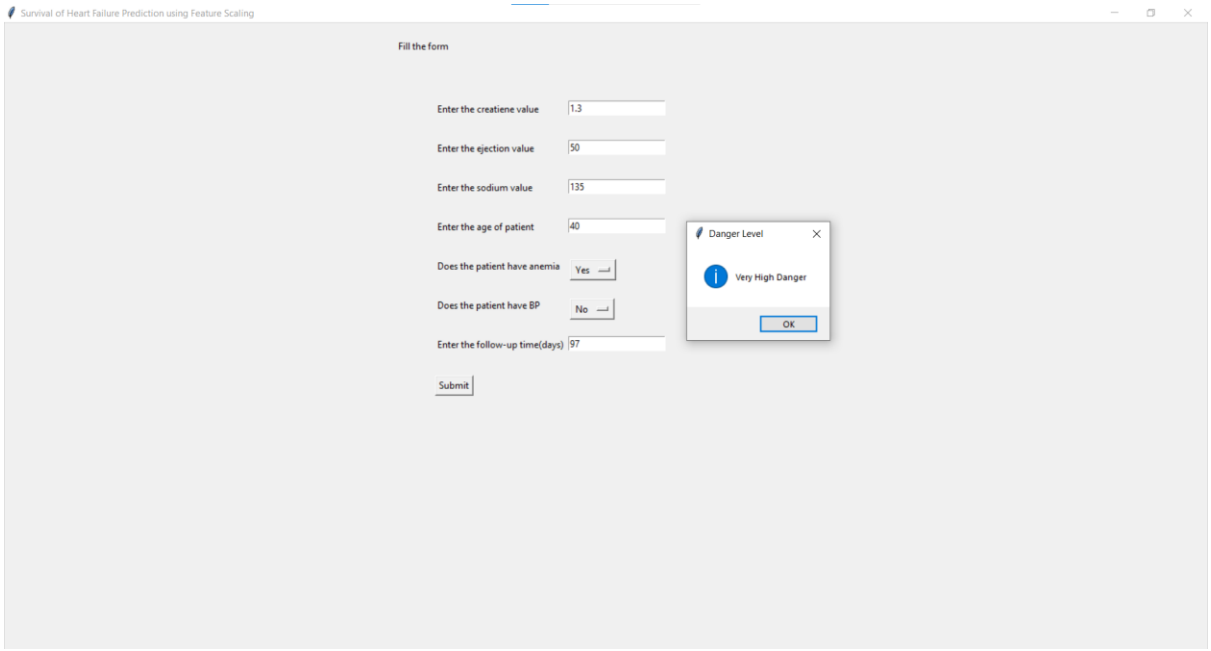

Fig 4.3.1 Test case showing very high danger level

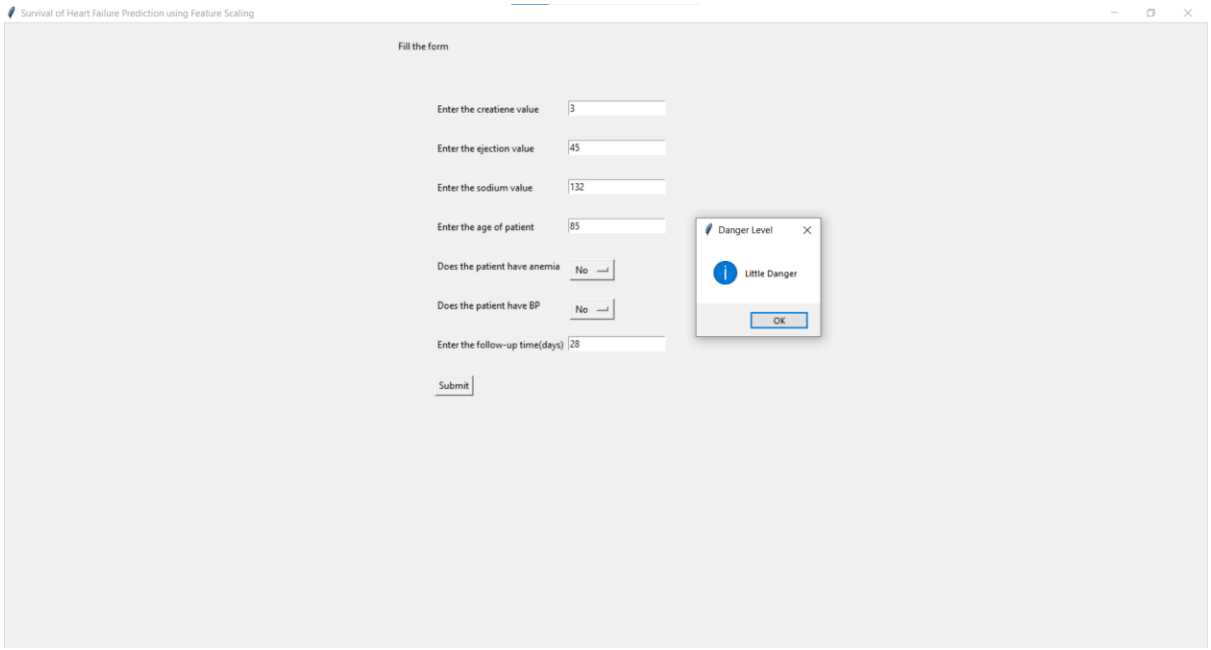

Fig 4.3.2 Test case showing little danger level

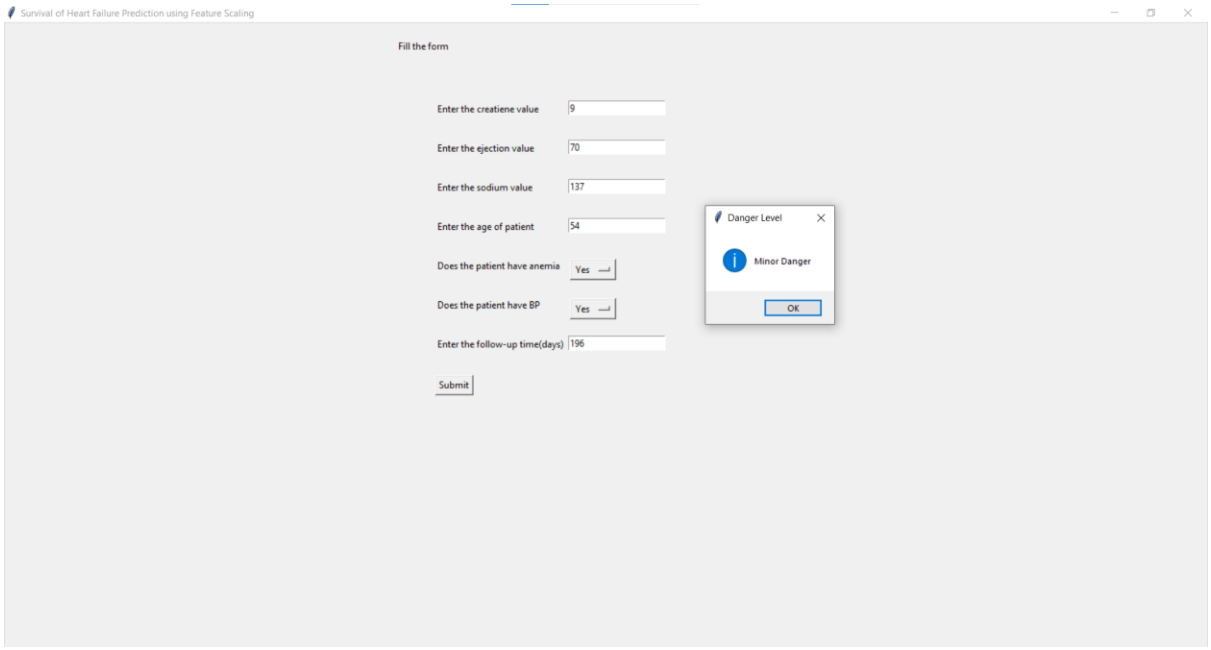

Fig 4.3.3 Test case showing minor danger level

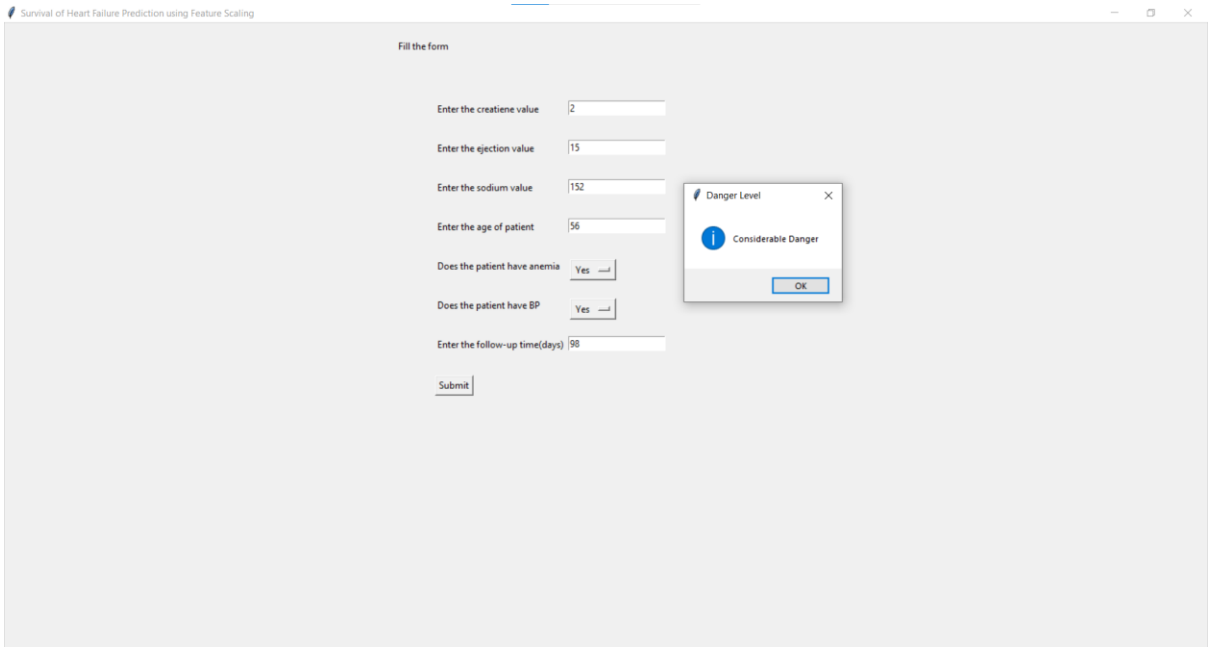

Fig 4.3.4 Test case showing considerable danger level

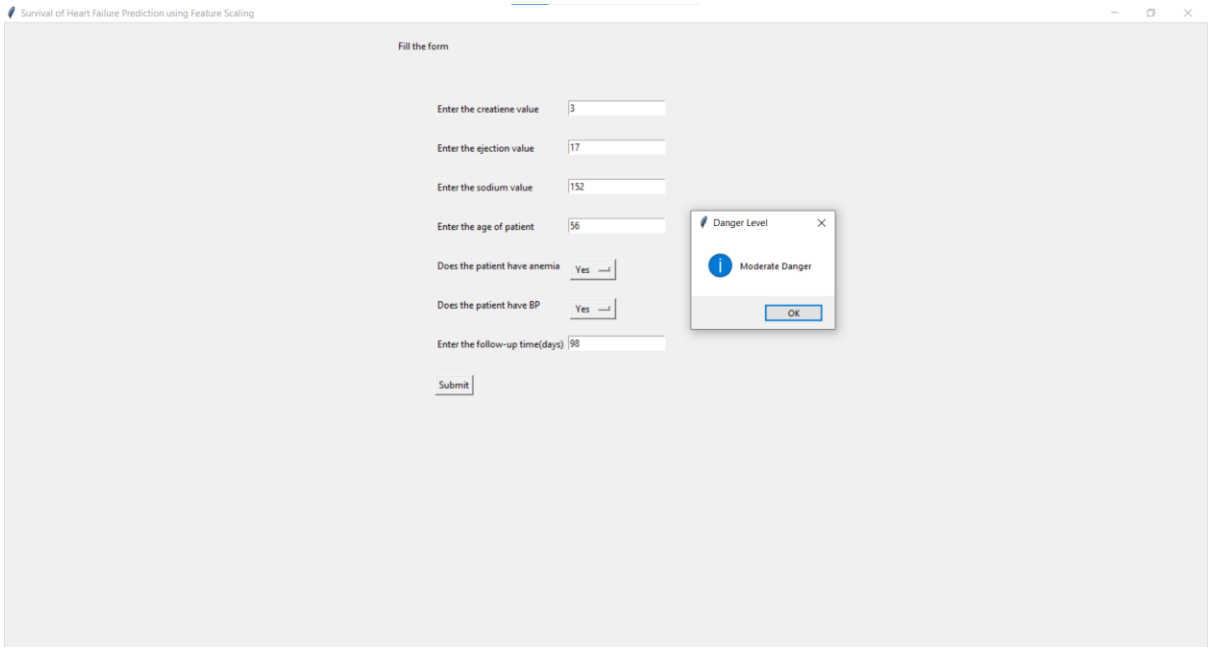

Fig 4.3.5 Test case showing moderate danger level

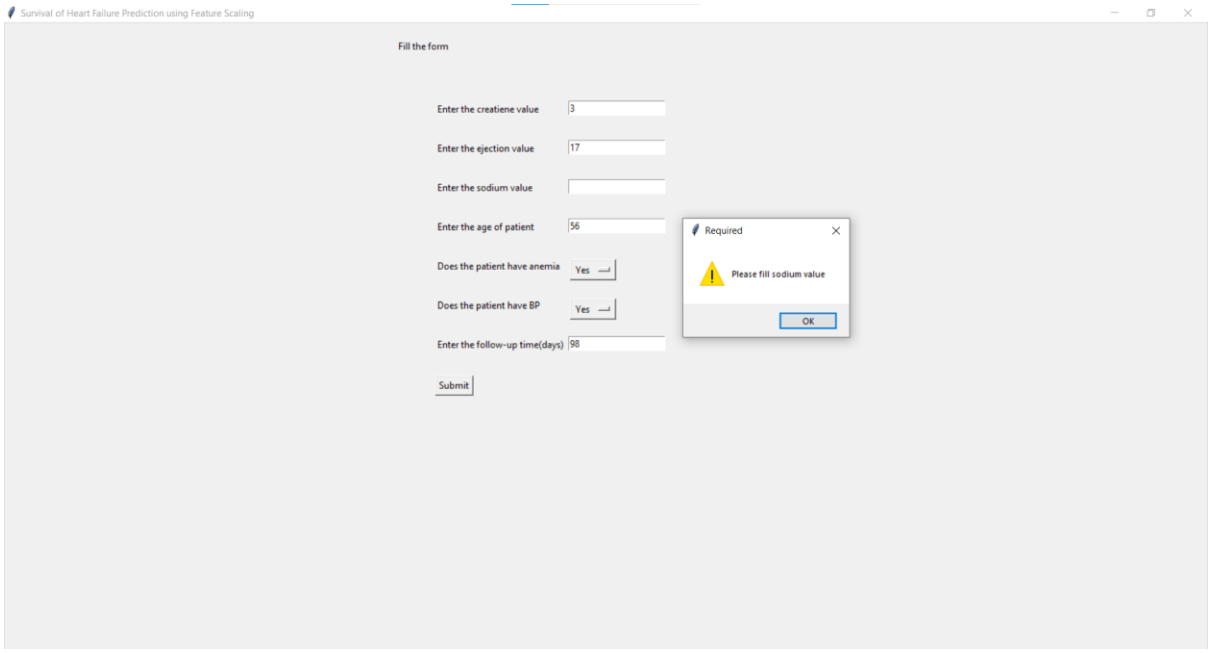

Fig 4.3.6 Test case showing warning to fill sodium field

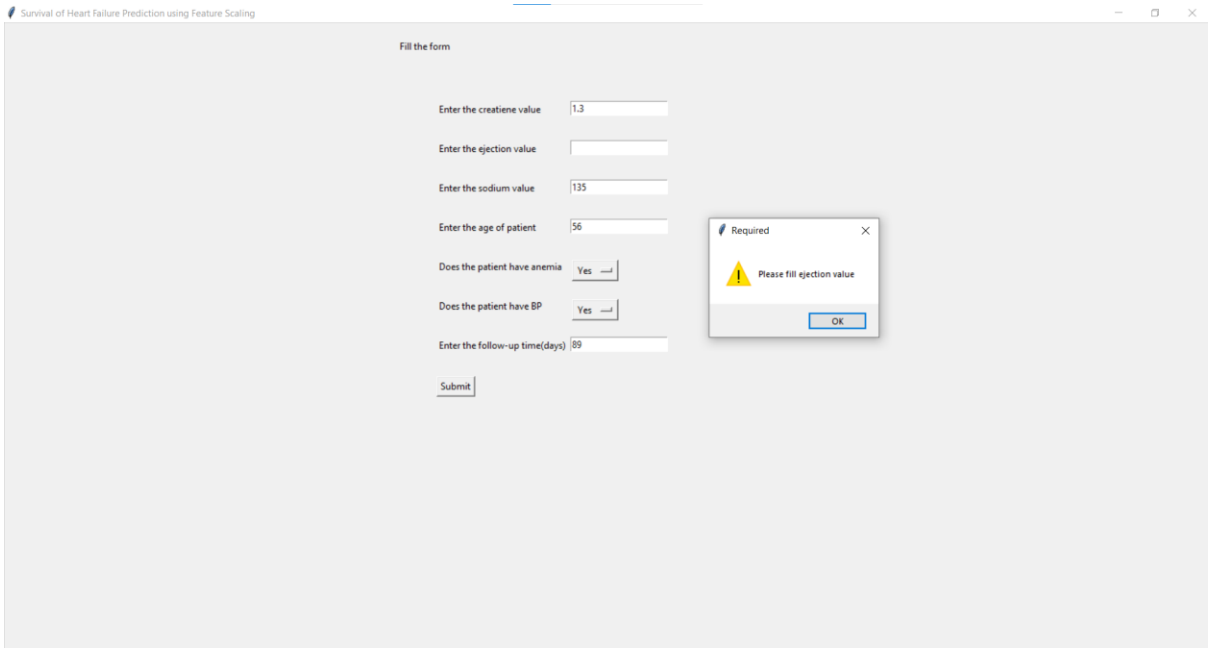

Fig 4.3.7 Test case showing warning to fill ejection field

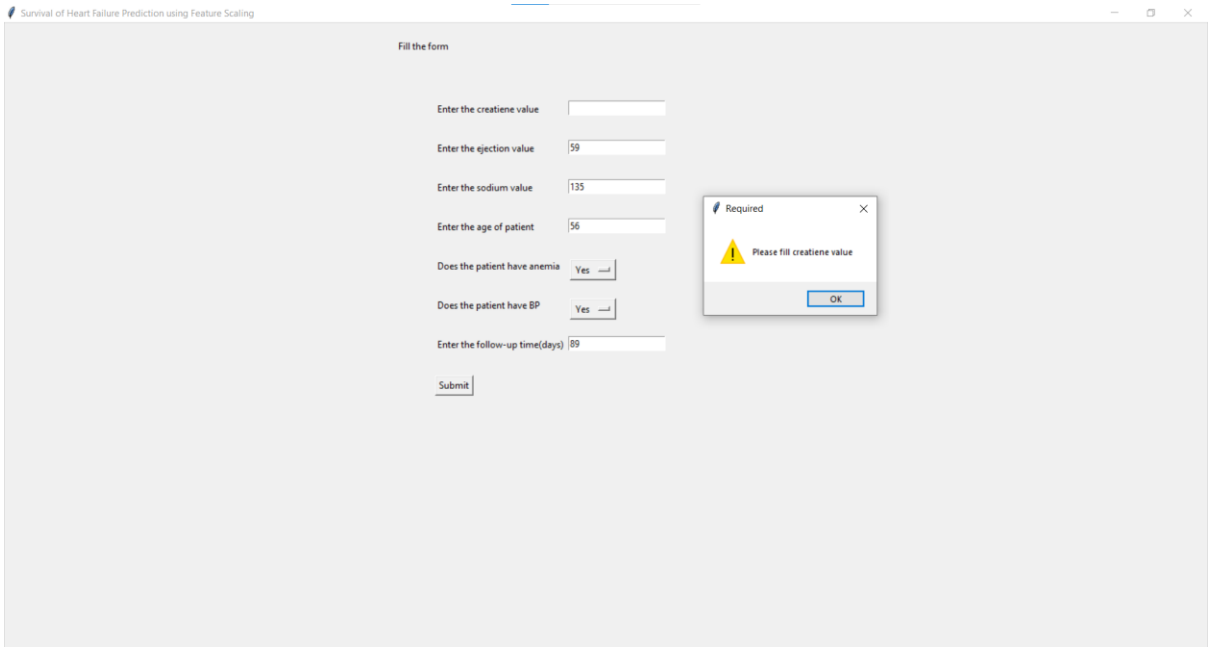

Fig 4.3.8 Test case showing warning to fill creatinine field

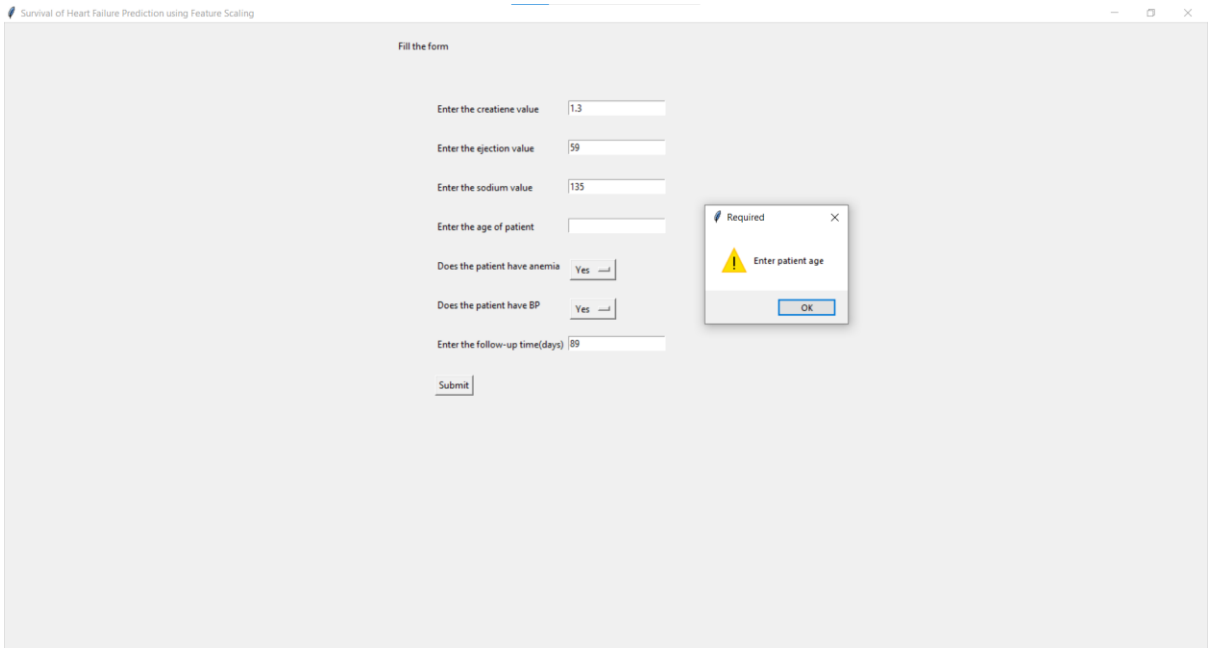

Fig 4.3.9 Test case showing warning to fill age field

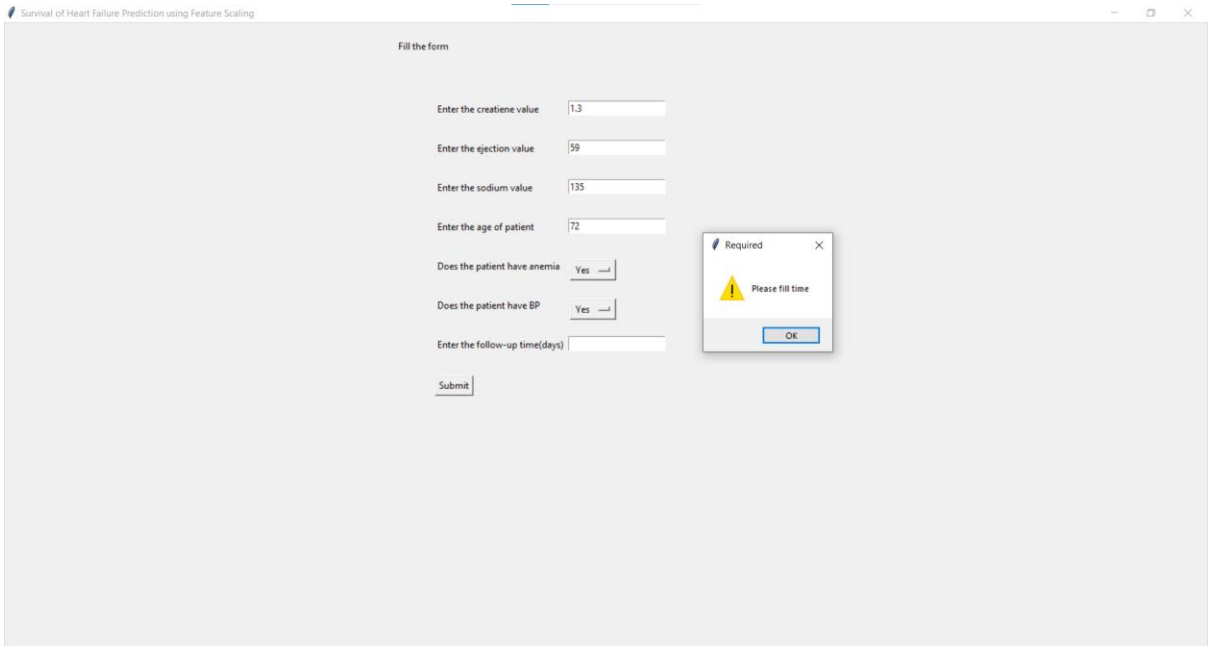

Fig 4.3.10 Test case showing warning to fill time field

CONCLUSION AND FUTURE SCOPE

After analysing the data and doing feature scaling, it was found that the features creatinine, sodium, ejection fraction, age, BP, anaemia and follow-up time are most relevant features when trying to predict the danger level of the patient. The danger level was classified into 6 classes namely, very high danger, high danger, considerable danger, moderate danger, minor danger and little danger by using machine learning algorithms. This discovery has the potential to impact on clinical practice, becoming a new supporting tool for physicians when predicting if a heart failure patient will survive or not.

In the future, it can use additional information about the physical features of the patient such as height, weight, body mass index etc and then improve the model further.

6. REFERENCES

- Davide Chicco and Giuseppe Jurman "Machine Learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone", BMC Medical Informatics and Decision Making, 3 February, 2020 pp.
- Prasanta Kumar Sahoo and Pravalika Jeripothula "Heart Failure Prediction using Machine Learning Techniques", 29 January, 2021 pp.
- Dataset https://www.kaggle.com/andrewmvd/heart-failure-clinical-data