PROJECT REPORT ON

Cardiovascular Disease Prediction Using Machine Learning Techniques

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

Submitted by

122003223 - SHAIK SUMAIYA - CSE
123003036 - BHAVANI OMKARINI M.J - CSE
123003233 - SIVANVITA NAIDU G.N - 123003233

Under the Guidance of

Dr. Renuga Devi, Assistant Professor-II

School of Computing

SASTRA DEEMED TO BE UNIVERSITY
(A University established under section 3 of the UGC Act, 1956)
Tirumalaisamudram
Thanjavur - 613401

SHANMUGHA
ARTS, SCIENCE, TECHNOLOGY & RESEARCH ACADEMY
(SASTRA DEEMED TO BE UNIVERSITY)

(A University Established under section 3 of the UGC Act, 1956)
TIRUMALAISAMUDRAM, THANJAVUR – 613401

BONAFIDE CERTIFICATE

This is to certify that the report titled "**Cardiovascular Disease Prediction Using Machine Learning Techniques"** submitted as a requirement for the course, CSE300: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by BHAVANI OMKARINI M.J (123003036, CSE), SHAIK SUMAIYA (123003223, CSE), SIVANVITA NAIDU G.N(123003233, CSE) during the academic year 2021-22, in the School of Computing,

Signature of Project Supervisor:

Name with Affiliation: Dr. T. Renugadevi-Assistant Professor-CSE/SoC

Date:29/6/22

Submitted for Mini Project Viva Voce held on 29/6/22

Examiner – I **Examiner – II**

ACKNOWLEDGEMENTS

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan,** Dean**,** Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli,** Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information & Communication Technology and **Dr. B.Santhi**, Associate Dean, Department of Computer Application.

Our guide **Dr.Renuga Devi**,Assistant Professor, School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through project.

LIST OF FIGRURES

ABBREIVIATIONS

- SMOTE – Synthetic Minority Over-sampling Technique
- KNN - K Nearest Neighbour

ABSTRACT

Nowadays, cardiovascular diseases are raising many concerns as they have become very common with the highest mortality rate. Some of the factors that cause cardiovascular diseases include age, stress, high blood pressure, cholesterol, family history, Body Mass Index (BMI), gender, and an unhealthy lifestyle. There are many proposed frameworks to predict diseases related to the heart using the above-mentioned factors. Here, in this article, we are going to propose a reliable and real environment framework for the early diagnosis and effective prediction of the diseases related to the heart with high accuracy and precision than the existing proposed frameworks. The validation of this framework is performed through a dataset namely Framingham with 99.1% accuracy. The proposed framework first handles missing values (using Mean Replacement technique) and deals with data imbalance (using SMOTE) then it deals with feature selection (via feature importance) and lastly, it uses an ensemble. In ensemble, they've used the combination of Logistic Regression and KNN classifiers in order to predict it with more accuracy.

KEY WORDS

- Machine Learning Techniques
- Mean Replacement technique
- SMOTE (Synthetic Minority Over-sampling Technique)
- Logistic Regression
- K-Nearest Neighbour (KNN)
- Ensemble

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In this modern world, where people are busy submerged in their daily schedules, which is leading to unhealthy life style in-turn causing stress, depression and anxiety. This is creating a tendency to resort excessive drinking, smoking and unhealthy habits such as taking drugs. All these unhealthy habits are the root cause for various heart related diseases. According to a report by WHO, the highest number of deaths in the world are happening due to Cardiovascular-Diseases.

The term Cardiovascular-Diseases (CVDs) is used to describe the heart related diseases. In general, there are four different types of Cardiovascular-Diseases which are:

- Coronary Heart Disease
- Transient ischemic attack (Stroke)
- Peripheral arterial disease
- Aortic disease

The major factors which contribute for Cardiovascular-Diseases include:

- High blood pressure
- Smoking
- Diabetes
- Body Mass Index (BMI)
- Cholesterol
- Age
- Family history
- Gender
- Stress
- Unhealthy life-style

The major challenge we have is to timely predict these Cardiovascular-Diseases with high accuracy so that we can reduce its mortality rate by medication and other measures.

During past few decades, many algorithms have been proposed to predict the heart diseases with accuracy using different techniques for prediction and different data sets.
The common data sets used for the prediction of Cardiovascular-Diseases include:

- Framingham data set
- Cleveland data set
- Heart disease data set
- Cardiovascular disease data set

Here, in this paper we use Framingham data set and in addition to it to verify the accuracy, we use Cleveland and Heart disease data sets.

There are many algorithms that are proposed and used for the accurate prediction of Cardiovascular-diseases. Here in this paper, we use:

- Logistic Regression
- K Nearest Neighbour (KNN)
- Ensemble (of both Logistic Regression and KNN)

CHAPTER 2

SUMMARY OF THE BASE PAPER

Title: An Integrated Machine Learning Framework for Effective Prediction of Cardiovascular Diseases
Published in: IEEE Access

Year: August 5, 2021

Base Paper URL: **https://ieeexplore.ieee.org/document/9491140/**

The base paper above has employed a four stage strategy which are Data Collection, Data Preprocessing, Feature Selection and Ensemble Classification **.** In Data Preprocessing, Removal of Outliers using Boxplot, Handling Missing Values by Mean Substitution and Data Balancing using SMOTE is done. After Data Preprocessing Feature Selection is done using Feature Importance Technique. Lastly an ensemble of Logistic and KNN is done for getting better accuracy

In base paper, Logistic and KNN are used separately before applying ensemble algorithm.After performing this a conclusion was made that ensemble algorithm gave more accuracy compared to Logistic and KNN

CHAPTER 3

WORKFLOW

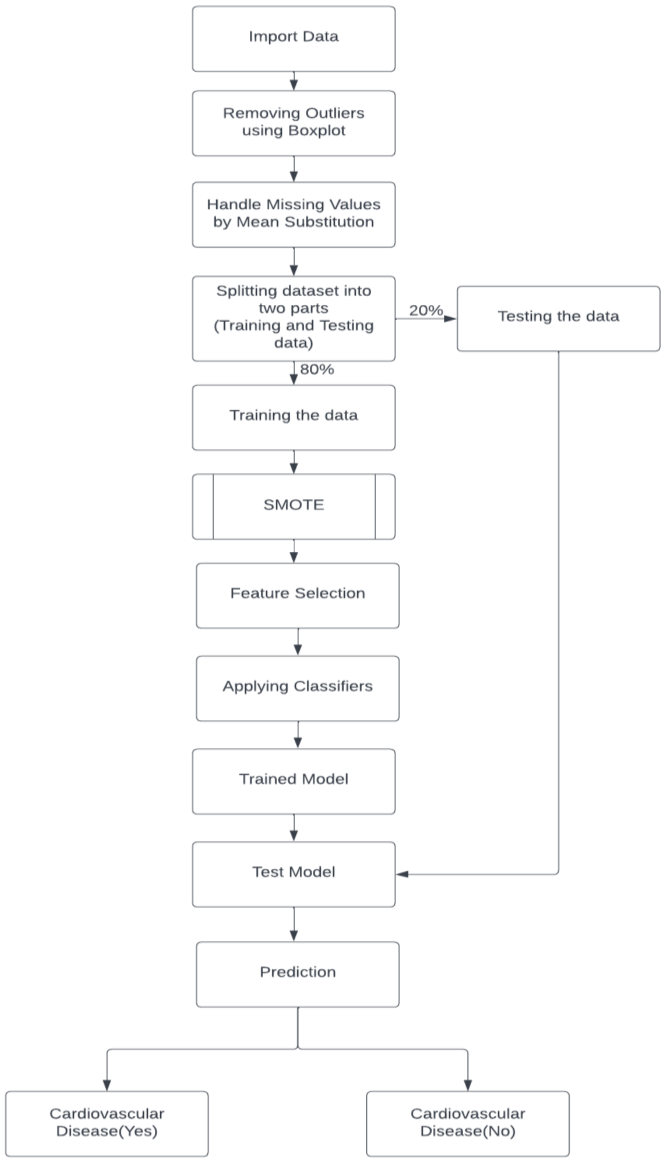


Fig 1: proposed Framework

CHAPTER 4

FRAMEWORK USED FOR THIS PROJECT

The aim of this project is to get best and high accuracy results with less number of features and less computational complexity.

1)DATASET:

<u>Main Data set:</u>

Framingham Dataset:

This dataset is the main dataset. This data set consists of 16 attributes. Target variable of this dataset is Ten-year CHD

<u>Data sets used for Validation:</u>

Heart Disease Dataset:

This dataset is used for the validation of the framework. This data set consists of 14 attributes. Target variable of this dataset is Target

Cleveland Dataset:

This dataset is used for the validation of the framework. This data set consists of 14 attributes. Target variable of this dataset is Condition

2)DATA PREPROCESSING:

2.1 Removal of Outliers using Boxplot

Initially in our framework, Data Pre-Processing is done where Outliers are removed. Outliers are removed using Boxplot. In boxplot there are first quartile and third quartile. Interquartile range is calculated by subtracting third quartile with first quartile. In this we calculate lower bound and upper bound where lowerbound=q1-1.5*IQR and upperbound=q1+1.5*IQR. If the attributes's values are less than lower bound and greater than upper bound then those values are considered as Outliers.In this way Outliers are removed using Boxplot

2.2 Handling Missing Values by Mean Substitution

For Handling Missing values, Our Framework uses Mean Substitution. In this values which are null is replaced by it's respective column's mean.

2.3 Data Balancing using SMOTE

Our Dataset consists of Imbalance Data. Balancing the data, Our Framework uses SMOTE. In SMOTE, the minority classes will be populated equal to majority classes and the data will be balanced. Balancing of Data improves accuarcy to greater extent

3)Feature Selection

Feature Importance Technique is used for Feature Selection in .Feature Importance has inbuilt tree based classifiers. Feature importance depends on the value of node probability.

Node Probability=No. of samples reaching that node/Total number of samples

If value of node is high then the feature is more important. In our framework features having score greater than 100 is considered as important feature.

4)Ensemble Classification

Even before applying ensemble, we applied Logistic Regression and KNN algorithm.

After this, our framework uses Ensemble Classification for getting high accuracy

CHAPTER 5

SOURCE CODE

DATA SET-1: FRAMINGHAM DATASET

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
data=pd.read_csv('framingham.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	Sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose
0	male	39	4.0	No	0.0	0.0	0	0	No	195.0	106.0	70.0	26.97	80.0	77.(
1	female	46	2.0	No	0.0	0.0	0	0	No	250.0	121.0	81.0	28.73	95.0	76.(
2	male	48	1.0	Yes	20.0	0.0	0	0	No	245.0	127.5	80.0	25.34	75.0	70.(
3	female	61	3.0	Yes	30.0	0.0	0	1	No	225.0	150.0	95.0	28.58	65.0	103.(
4	female	46	3.0	Yes	23.0	0.0	0	0	No	285.0	130.0	84.0	23.10	85.0	85.(

In [7]:
```
dummy1=pd.get_dummies(data['Sex'])
```

In [8]:
```
dummy1.head()
```

Out[8]:

	female	male
0	0	1
1	1	0
2	0	1
3	1	0
4	1	0

In [9]:
```
data2=pd.concat((data,dummy1),axis=1)
```

In [10]:
```
data2.head()
```

Out[10]:

	Sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose
0	male	39	4.0	No	0.0	0.0	0	0	No	195.0	106.0	70.0	26.97	80.0	77.(
1	female	46	2.0	No	0.0	0.0	0	0	No	250.0	121.0	81.0	28.73	95.0	76.(
2	male	48	1.0	Yes	20.0	0.0	0	0	No	245.0	127.5	80.0	25.34	75.0	70.(
3	female	61	3.0	Yes	30.0	0.0	0	1	No	225.0	150.0	95.0	28.58	65.0	103.(
4	female	46	3.0	Yes	23.0	0.0	0	0	No	285.0	130.0	84.0	23.10	85.0	85.(

In [11]:
```
data2=data2.drop(['Sex'],axis=1)
```

In [12]:
```
data2=data2.drop(['female'],axis=1)
```

In [17]:
```
dummy2=pd.get_dummies(data['currentSmoker'])
```

In [18]:
```
dummy2.head()
```

Out[18]:

	No	Yes
0	1	0
1	1	0
2	0	1
3	0	1
4	0	1

In [19]:
```
data=pd.concat((data2,dummy2),axis=1)
```

In [20]:
```
data.head()
```

Out[20]:

	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYe
0	39	4.0	No	0.0	0.0	0	0	No	195.0	106.0	70.0	26.97	80.0	77.0	
1	46	2.0	No	0.0	0.0	0	0	No	250.0	121.0	81.0	28.73	95.0	76.0	
2	48	1.0	Yes	20.0	0.0	0	0	No	245.0	127.5	80.0	25.34	75.0	70.0	
3	61	3.0	Yes	30.0	0.0	0	1	No	225.0	150.0	95.0	28.58	65.0	103.0	
4	46	3.0	Yes	23.0	0.0	0	0	No	285.0	130.0	84.0	23.10	85.0	85.0	

In [21]:
```
data=data.drop(['No'],axis=1)
```

In [22]:
```
data=data.drop(['currentSmoker'],axis=1)
```

In [23]:
```
data=data.rename(columns={"Yes":"currentSmoker"})
```

In [25]:
```
dummy3=pd.get_dummies(data['diabetes'])
```

In [26]:
```
data=pd.concat([data,dummy3],axis=1)
```

In [27]:
```
data.head()
```

Out[27]:

	age	education	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD	Sex	cu
0	39	4.0	0.0	0.0	0	0	No	195.0	106.0	70.0	26.97	80.0	77.0	0	1	
1	46	2.0	0.0	0.0	0	0	No	250.0	121.0	81.0	28.73	95.0	76.0	0	0	
2	48	1.0	20.0	0.0	0	0	No	245.0	127.5	80.0	25.34	75.0	70.0	0	1	
3	61	3.0	30.0	0.0	0	1	No	225.0	150.0	95.0	28.58	65.0	103.0	1	0	
4	46	3.0	23.0	0.0	0	0	No	285.0	130.0	84.0	23.10	85.0	85.0	0	0	

In [28]:
```
data=data.drop(['No'],axis=1)
```

In [29]:
```
data=data.drop(['diabetes'],axis=1)
```

In [30]:
```
data=data.rename(columns={'Yes':'diabetes'})
```

Handling Missing Values

In [34]:
```
data.isnull().sum()
```

Out[34]:
```
age                 0
education         105
cigsPerDay         29
BPMeds             53
prevalentStroke     0
prevalentHyp        0
totChol            50
sysBP               0
diaBP               0
BMI                19
heartRate           1
glucose           388
TenYearCHD          0
Sex                 0
currentSmoker       0
diabetes            0
dtype: int64
```

In [35]:
```
data['education'].replace(np.NaN,data['education'].mean(),inplace=True)
```

In [37]:
```
data['cigsPerDay'].replace(np.NaN,data['cigsPerDay'].mean(),inplace=True)
```

In [38]:
```
data['BPMeds'].replace(np.NaN,data['BPMeds'].mean(),inplace=True)
```

In [39]:
```
data['totChol'].replace(np.NaN,data['totChol'].mean(),inplace=True)
```

In [40]:
```
data['BMI'].replace(np.NaN,data['BMI'].mean(),inplace=True)
```

In [41]:
```
data['glucose'].replace(np.NaN,data['glucose'].mean(),inplace=True)
```

In [42]:
```
data['heartRate'].replace(np.NaN,data['heartRate'].mean(),inplace=True)
```

In [43]:
```
data.isnull().sum()
```

Out[43]:
```
age                0
education          0
cigsPerDay         0
BPMeds             0
prevalentStroke    0
prevalentHyp       0
totChol            0
sysBP              0
diaBP              0
BMI                0
heartRate          0
glucose            0
TenYearCHD         0
Sex                0
currentSmoker      0
diabetes           0
dtype: int64
```

Data Balancing using Synthetic Minority Over-sampling Technique(SMOTE)

In [45]:
```
data.TenYearCHD.value_counts()
```

Out[45]:
```
0    3596
1     644
Name: TenYearCHD, dtype: int64
```

In [46]:
```
#x=data[data.columns.difference(['TenYearCHD'])]
x=data.drop(['TenYearCHD'], axis = 1)
y=data.TenYearCHD
x.head()
```

Out[46]:

	age	education	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	totChol	sysBP	diaBP	BMI	heartRate	glucose	Sex	currentSmoker	diabetes
0	39	4.0	0.0	0.0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	1	0	0
1	46	2.0	0.0	0.0	0	0	250.0	121.0	81.0	28.73	95.0	76.0	0	0	0
2	48	1.0	20.0	0.0	0	0	245.0	127.5	80.0	25.34	75.0	70.0	1	1	0
3	61	3.0	30.0	0.0	0	1	225.0	150.0	95.0	28.58	65.0	103.0	0	1	0
4	46	3.0	23.0	0.0	0	0	285.0	130.0	84.0	23.10	85.0	85.0	0	1	0

In [47]:
```
from sklearn.model_selection import train_test_split
```

In [48]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
print (x_train.shape, y_train.shape)
print (x_test.shape, y_test.shape)
```

```
(3392, 15) (3392,)
(848, 15) (848,)
```

```
In [49]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [50]: model=KNeighborsClassifier()
```

```
In [51]: model.fit(x_train,y_train)
         y_predict=model.predict(x_test)
```

```
In [52]: pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\user\anaconda3\lib\site-packages (0.0)Note: you may need to restart the
kernel to use updated packages.
Requirement already satisfied: imbalanced-learn in c:\users\user\anaconda3\lib\site-packages (from imblearn) (0.9.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn->imb
learn) (2.1.0)
Requirement already satisfied: scikit-learn>=1.0.1 in c:\users\user\appdata\roaming\python\python38\site-packages (from imba
lanced-learn->imblearn) (1.0.2)
Requirement already satisfied: joblib>=0.11 in c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.0.1)
Requirement already satisfied: numpy>=1.14.6 in c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.20.1)
Requirement already satisfied: scipy>=1.1.0 in c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.6.2)
```

```
In [53]: from sklearn.metrics import accuracy_score
         print(accuracy_score(y_test,y_predict))
         pd.crosstab(y_test,y_predict)
```

```
0.8290094339622641
```

```
In [54]: pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\user\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\user\anaconda3\lib\site-packages (from imblearn) (0.9.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn->imb
learn) (2.1.0)
Requirement already satisfied: scikit-learn>=1.0.1 in c:\users\user\appdata\roaming\python\python38\site-packages (from imba
lanced-learn->imblearn) (1.0.2)
Requirement already satisfied: numpy>=1.14.6 in c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.20.1)
Requirement already satisfied: scipy>=1.1.0 in c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.6.2)
Requirement already satisfied: joblib>=0.11 in c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.0.1)
Note: you may need to restart the kernel to use updated packages.
```

```
In [55]: from imblearn.over_sampling import SMOTE
         smote=SMOTE()
```

```
In [56]: x_smote_train,y_smote_train=smote.fit_resample(x_train.astype('float'),y_train)
```

```
In [57]: from collections import Counter
         print("Before applying SMOTE:",Counter(y_train))
         print("After applying SMOTE:",Counter(y_smote_train))
```

```
Before applying SMOTE: Counter({0: 2884, 1: 508})
After applying SMOTE: Counter({0: 2884, 1: 2884})
```

Feature Selection

```
In [59]: from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import chi2
```

```
In [60]: important_features = SelectKBest(score_func=chi2, k=15)
         fit = important_features.fit(x,y)
         data_scores = pd.DataFrame(fit.scores_)
         data_columns = pd.DataFrame(x.columns)
         feature_Scores = pd.concat([data_columns,data_scores],axis=1)
         feature_Scores.columns = ['Attributes','Score']
```

In [61]:
```
featureScores = feature_Scores.sort_values(by='Score', ascending=False)
featureScores
```

Out[61]:

	Attributes	Score
7	sysBP	727.935535
11	glucose	391.151105
0	age	319.266019
6	totChol	235.502392
2	cigsPerDay	220.812679
8	diaBP	152.748563
5	prevalentHyp	92.048736
14	diabetes	39.144944
3	BPMeds	30.615014
12	Sex	18.899930
4	prevalentStroke	16.109887
9	BMI	15.227367
1	education	6.233112
10	heartRate	4.232372
13	currentSmoker	0.811334

In [62]:
```
data=data[['sysBP','age','totChol','cigsPerDay','diaBP','TenYearCHD']]
```

In [63]:
```
data.head()
```

Out[63]:

	sysBP	age	totChol	cigsPerDay	diaBP	TenYearCHD
0	106.0	39	195.0	0.0	70.0	0
1	121.0	46	250.0	0.0	81.0	0
2	127.5	48	245.0	20.0	80.0	0
3	150.0	61	225.0	30.0	95.0	1
4	130.0	46	285.0	23.0	84.0	0

In [64]:
```
from sklearn.model_selection import train_test_split

y = data['TenYearCHD'] #target variable
X = data.drop(['TenYearCHD'], axis = 1) #features
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

```
(3392, 5) (3392,)
(848, 5) (848,)
```

DATA SET-2: CLEVELAND DATASET

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: data = pd.read_csv('heart_cleveland_upload.csv')
```

```
In [3]: data.head()
```

Out[3]:

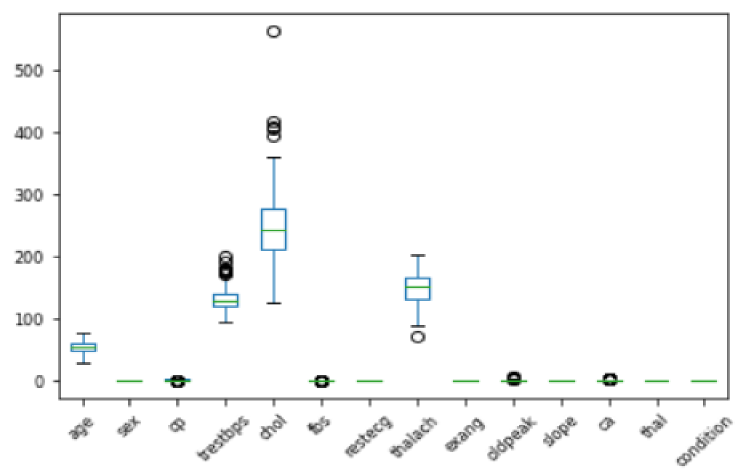
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	condition
0	69	1	0	160	234	1	2	131	0	0.1	1	1	0	0
1	69	0	0	140	239	0	0	151	0	1.8	0	2	0	0
2	66	0	0	150	226	0	0	114	0	2.6	2	0	0	0
3	65	1	0	138	282	1	2	174	0	1.4	1	1	0	1
4	64	1	0	110	211	0	2	144	1	1.8	1	0	0	0

Removal of Outliers using Boxplot

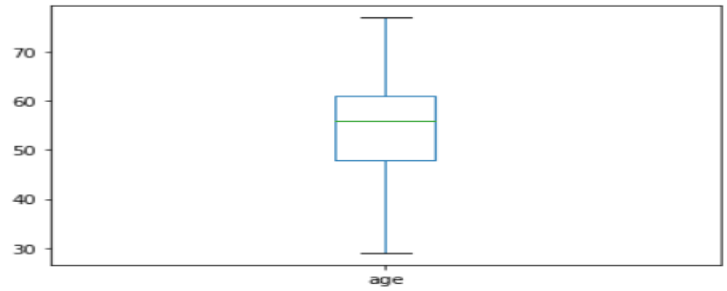
```
In [4]: data.shape
Out[4]: (297, 14)
```

```
In [5]: def  plot_boxplot(df,ft):
            df.boxplot(column=[ft])
            plt.grid(False)
            plt.show()
```

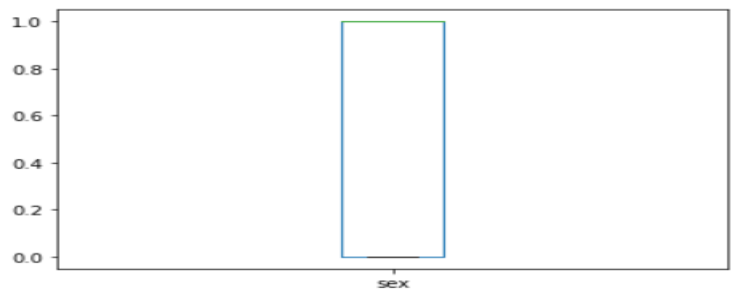
```
In [6]: data.boxplot(grid = False, rot = 45, fontsize = 8)
Out[6]: <AxesSubplot:>
```



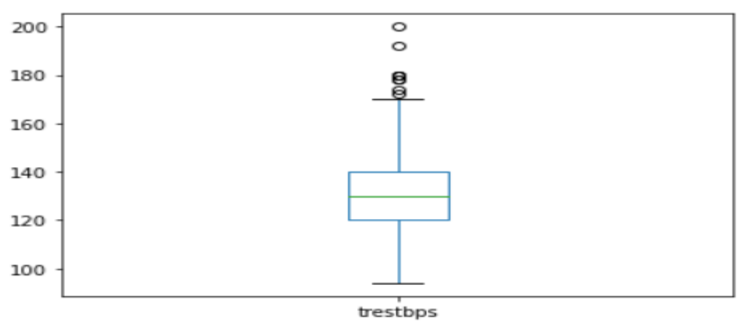
In [7]:
```
plot_boxplot(data,'age')
```



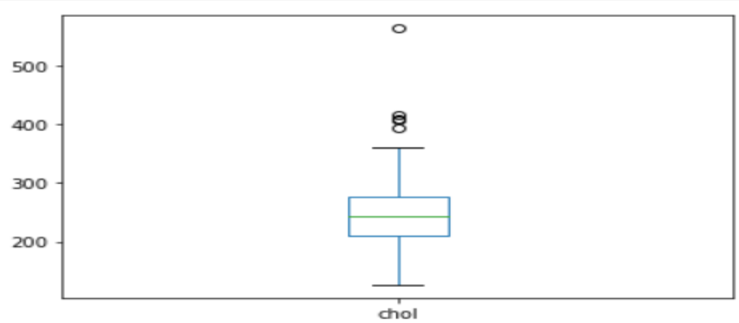
In [8]:
```
plot_boxplot(data,'sex')
```



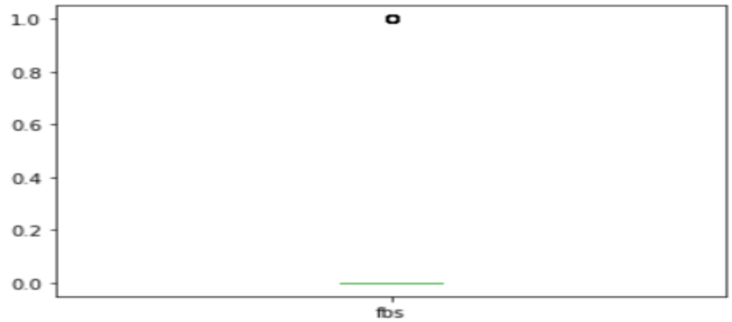
In [9]:
```
plot_boxplot(data,'trestbps')
```



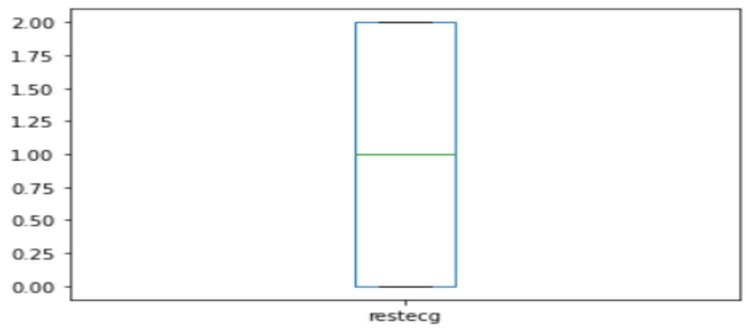
In [10]:
```
plot_boxplot(data,'chol')
```



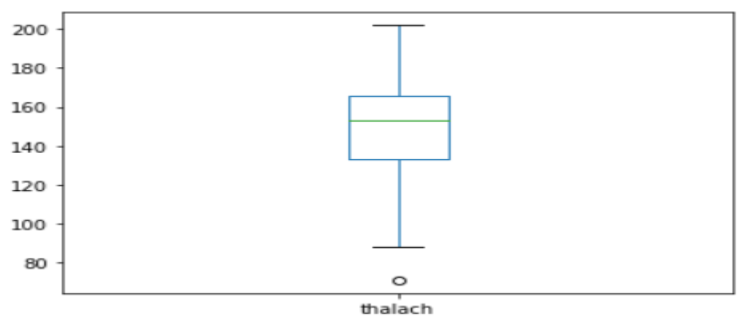
In [11]:
```
plot_boxplot(data,'fbs')
```



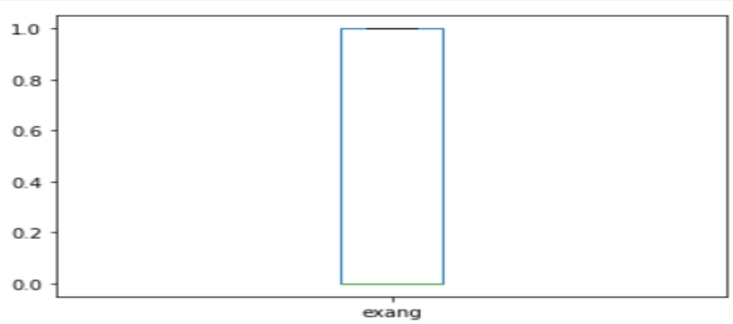
In [12]:
```
plot_boxplot(data,'restecg')
```



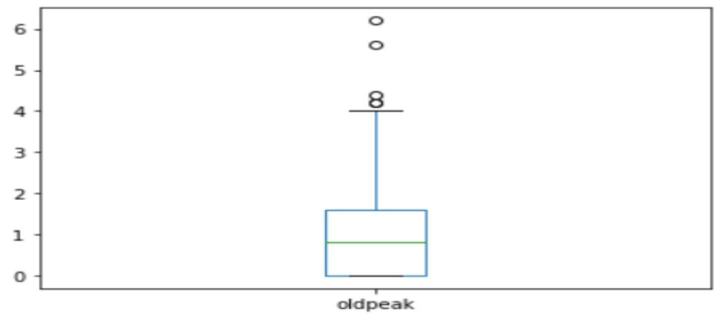
In [13]:
```
plot_boxplot(data,'thalach')
```



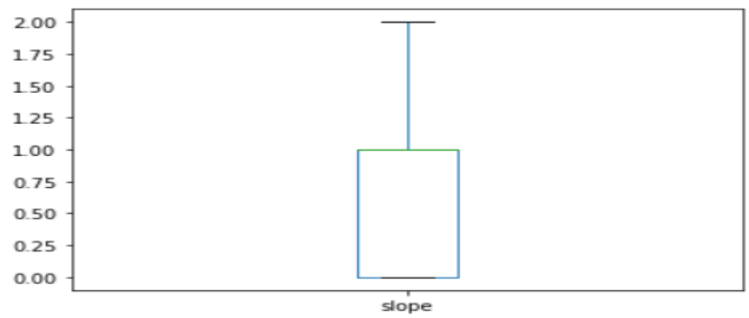
In [14]:
```
plot_boxplot(data,'exang')
```



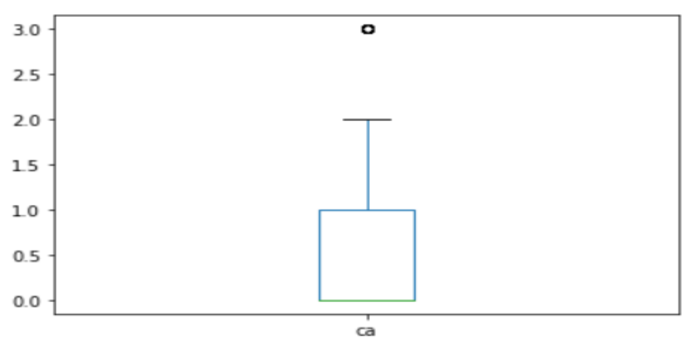
In [15]:
```
plot_boxplot(data,'oldpeak')
```



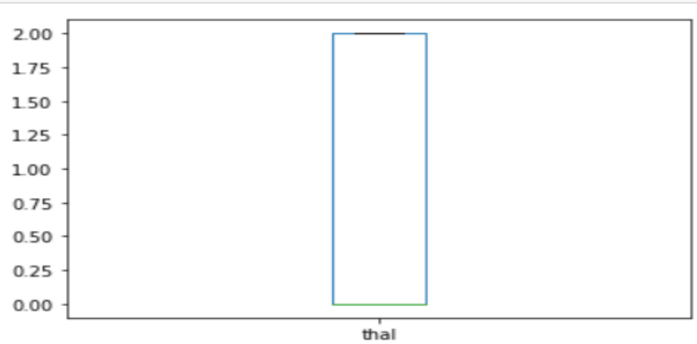
In [16]:
```
plot_boxplot(data,'slope')
```



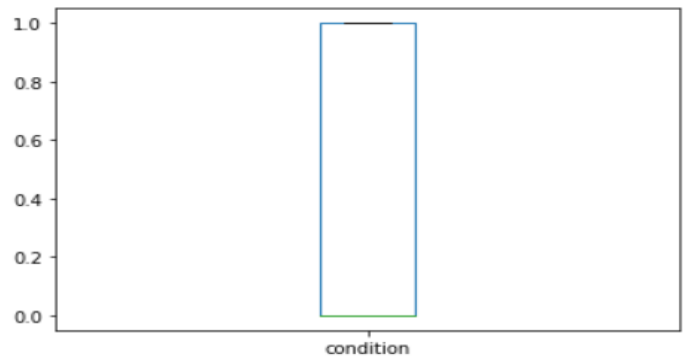
In [17]:
```
plot_boxplot(data,'ca')
```



In [18]:
```
plot_boxplot(data,'thal')
```



```
In [19]: plot_boxplot(data,'condition')
```




```
In [20]: def Outlier(df,ft):
             Q1 = df[ft].quantile(0.25)
             Q3 = df[ft].quantile(0.75)
             IQR = Q3-Q1
             lower_bound = Q1-1.5*IQR
             upper_bound = Q3+1.5*IQR
             lt = df.index[(df[ft]<lower_bound)|(df[ft]>upper_bound)]
             return lt
```

```
In [21]: index_list = []
         for feature in data.columns:
             index_list.extend(Outlier(data,feature))
```

```
In [22]: index_list

Out[22]: [0,
          1,
          2,
          3,
          4,
          5,
          6,
          7,
          8,
          9,
          10,
          11,
          12,
          13,
          14,
          15,
          16,
          17,
          18,
```

```
In [23]: def remove(df,lt):
             lt = sorted(set(lt))
             df = df.drop(lt)
             return df
```

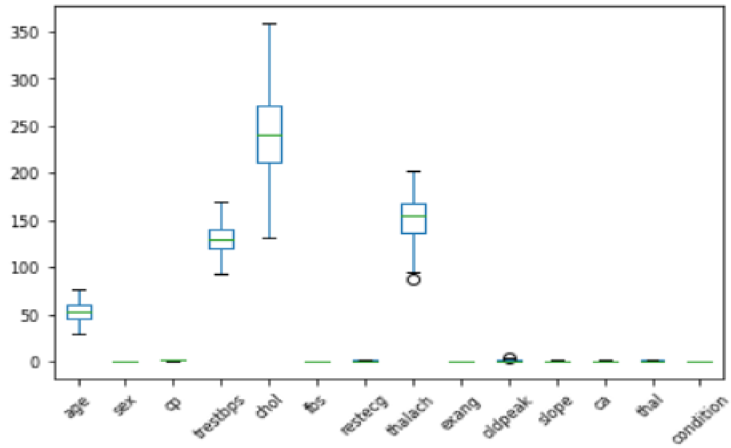
```
In [24]: data = remove(data,index_list)
```

```
In [25]: data.shape

Out[25]: (212, 14)
```

In [25]:
```
data.shape
```
Out[25]: (212, 14)

In [26]:
```
data.boxplot(grid = False, rot = 45, fontsize = 8)
```
Out[26]: <AxesSubplot:>



Handling Missing Values

In [27]:
```
data.isnull().sum()
```
Out[27]:
```
age          0
sex          0
cp           0
trestbps     0
chol         0
fbs          0
restecg      0
thalach      0
exang        0
oldpeak      0
slope        0
ca           0
thal         0
condition    0
dtype: int64
```

In [28]:
```
data.head()
```
Out[28]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	condition
23	74	0	1	120	269	0	2	121	1	0.2	0	1	0	0
24	71	0	1	160	302	0	0	162	0	0.4	0	2	0	0
25	70	1	1	156	245	0	2	143	0	0.0	0	0	0	0
27	63	0	1	140	195	0	0	179	0	0.0	0	2	0	0
28	62	1	1	120	281	0	2	103	0	1.4	1	1	2	1

Data Balancing using Synthetic Minority Over-sampling Tec (SMOTE)

In [29]:
```
data.condition.value_counts()
```
Out[29]:
```
0    121
1     91
Name: condition, dtype: int64
```

In [30]:
```
x = data.drop(['condition'],axis = 1)
y = data.condition
x.head()
```
Out[30]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
23	74	0	1	120	269	0	2	121	1	0.2	0	1	0
24	71	0	1	160	302	0	0	162	0	0.4	0	2	0
25	70	1	1	156	245	0	2	143	0	0.0	0	0	0
27	63	0	1	140	195	0	0	179	0	0.0	0	2	0
28	62	1	1	120	281	0	2	103	0	1.4	1	1	2

In [31]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2)
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```
```
(169, 13) (169,)
(43, 13) (43,)
```

In [33]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [34]:
```
model = KNeighborsClassifier()
```

In [35]:
```
model.fit(x_train, y_train)
y_predict = model.predict(x_test)
```

In [36]:
```
pip install imblearn
```
```
Requirement already satisfied: imblearn in c:\users\bhavani omkarini\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imblearn) (0.9.
0)
Requirement already satisfied: scipy>=1.1.0 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-learn->im
blearn) (1.7.1)
Requirement already satisfied: numpy>=1.14.6 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-learn->i
mblearn) (1.20.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-l
earn->imblearn) (2.2.0)
Requirement already satisfied: scikit-learn>=1.0.1 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-le
arn->imblearn) (1.0.2)
Requirement already satisfied: joblib>=0.11 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-learn->im
blearn) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [37]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_predict))
pd.crosstab(y_test, y_predict)
```
```
0.6511627906976745
```

Out[37]:

col_0	0	1
condition		
0	14	7
1	8	14

In [38]:
```
pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\bhavani omkarini\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imblearn) (0.9.
0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-l
earn->imblearn) (2.2.0)
Requirement already satisfied: scikit-learn>=1.0.1 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-le
arn->imblearn) (1.0.2)
Requirement already satisfied: joblib>=0.11 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-learn->im
blearn) (1.1.0)
Requirement already satisfied: numpy>=1.14.6 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-learn->i
mblearn) (1.20.3)
Requirement already satisfied: scipy>=1.1.0 in c:\users\bhavani omkarini\anaconda3\lib\site-packages (from imbalanced-learn->im
blearn) (1.7.1)
Note: you may need to restart the kernel to use updated packages.
```

In [39]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
```

In [40]:
```
x_smote_train, y_somte_train = smote.fit_resample(x_train.astype('float'), y_train)
```

In [41]:
```
from collections import Counter
print("Before applying SMOTE:", Counter(y_train))
print("After  applying SMOTE:", Counter(y_somte_train))
```

```
Before applying SMOTE: Counter({0: 100, 1: 69})
After  applying SMOTE: Counter({0: 100, 1: 100})
```

Feature Selection ¶

In [42]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [43]:
```
important_features = SelectKBest(score_func = chi2, k = 13)
fit = important_features.fit(x,y)
data_scores = pd.DataFrame(fit.scores_)
data_columns = pd.DataFrame(x.columns)
feature_scores = pd.concat([data_columns, data_scores], axis = 1)
feature_scores.columns = ['Attributes', 'Score']
```

In [44]:
```
featureScores = feature_scores.sort_values(by = 'Score', ascending = False)
featureScores
```

Out[44]:

	Attributes	Score
7	thalach	131.535824
12	thal	88.419616
9	oldpeak	52.772890
11	ca	47.394584
8	exang	28.556171
0	age	19.289654
10	slope	15.115813
2	cp	12.156599
1	sex	9.347546
6	restecg	7.413737
3	trestbps	4.650644
4	chol	3.712310
5	fbs	NaN

In [45]:
```python
data = data[['fbs']]
```

In [46]:
```python
data.head()
```

Out[46]:

	fbs
23	0
24	0
25	0
27	0
28	0

In [47]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn import datasets, linear_model
from imblearn.pipeline import Pipeline
from sklearn.svm import SVC
from imblearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold
# decision tree on imbalanced dataset with SMOTE oversampling and random undersampling
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from imblearn.pipeline import pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
import sklearn.linear_model as lm

#fit a model
lm = lm.LogisticRegression()
model = lm.fit(x_train, y_train)
over = SMOTE(sampling_strategy=0.1)
steps = [('over',over),('model',model)]
pipeline = Pipeline(steps = steps)
x,y = make_classification(n_samples = 10000, n_features = 5, n_redundant = 0, n_clusters_per_class = 1, weights = [0.99], flip_y
cv =RepeatedStratifiedKFold(n_splits = 5, n_repeats = 3, random_state = 1)
scores = cross_val_score(pipeline,x,y,scoring = 'roc_auc', cv =cv, n_jobs = -1)
#model.score(x_test, y_test)
print('Mean ROC AUC : %3f' % mean(scores))
```

```
C:\Users\Bhavani Omkarini\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Mean ROC AUC : 0.942093
```

In [48]:
```
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

kfold = model_selection.KFold(n_splits = 5)

#create the sub models
estimators = []
model1 = LogisticRegression().fit(x_test, y_test)
estimators.append(('logistic', model1))
model2 = KNeighborsClassifier(n_neighbors = 3)
estimators.append(('cart',model2))

#create the ensemble model
ensemble = VotingClassifier(estimators)

over = SMOTE(sampling_strategy = 0.1)
steps = [('over', over), ('model',ensemble)]
pipeline = Pipeline(steps = steps)

results = model_selection.cross_val_score(pipeline, x, y, cv = kfold)
print(results.mean())
```

```
C:\Users\Bhavani Omkarini\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
0.9915
```

In [50]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import datasets, linear_model
from imblearn.pipeline import Pipeline
from sklearn.svm import SVC
from imblearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold
# decision tree  on imbalanced dataset with SMOTE oversampling and random undersampling
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
import sklearn.linear_model as lm
```

```
# fit a model
lm = lm.LogisticRegression()
model = lm.fit(x_train, y_train)
over = SMOTE(sampling_strategy=0.1)
steps = [('over', over), ('model', model)]
pipeline = Pipeline(steps=steps)
X, y = make_classification(n_samples=10000, n_features=5, n_redundant=0,
n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
#model.score(X_test, y_test)
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
C:\Users\Bhavani Omkarini\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Mean ROC AUC: 0.943
```

In [52]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
from sklearn.metrics import classification_report, confusion_matrix
#print(confusion_matrix(y_test, y_pred))
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred, normalize=True)
)
```

```
0.7674418604651163
```

In [53]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred, normalize=True)
)
```

```
0.6511627906976745
```

DATA SET-3: HEART DISEASE DATASET

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
[ ] from google.colab import files
    uploaded=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving heart.csv to heart (2).csv
```

```
[ ] data=pd.read_csv('heart.csv')
```

```
[ ] data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

```
[ ] data.tail()
```

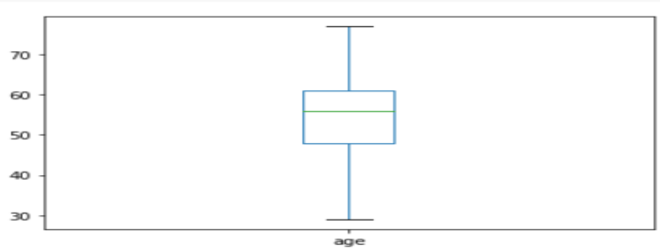
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

REMOVAL OF OUTLIERS USING BOXPLOT

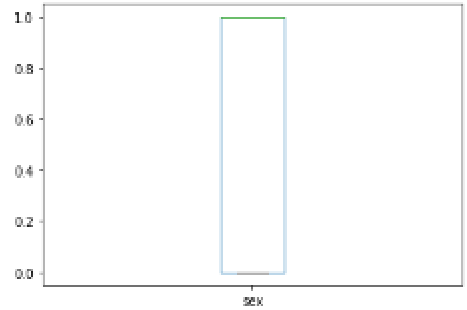
```
[ ] data.shape
    (1025, 14)
```

```
[ ] def plot_boxplot(df,ft):
      df.boxplot(column=[ft])
      plt.grid(False)
      plt.show()
```

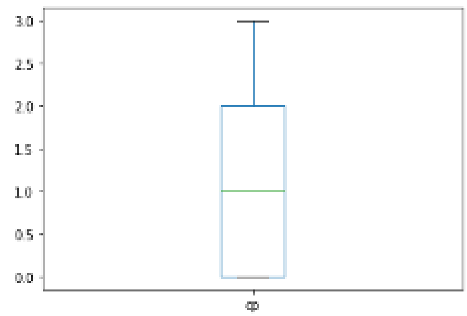
```
[ ] plot_boxplot(data,'age')
```



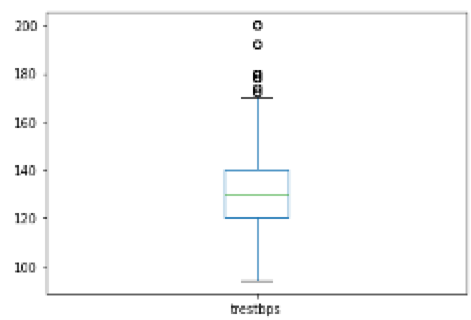
```
[ ] plot_boxplot(data,'sex')
```



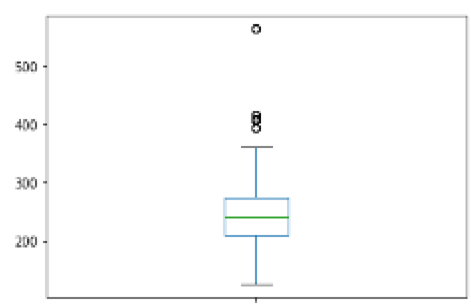

```
[ ] plot_boxplot(data,'cp')
```



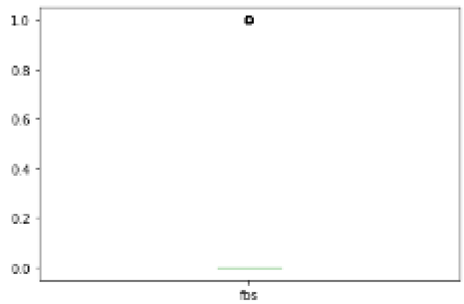

```
[ ] plot_boxplot(data,'trestbps')
```



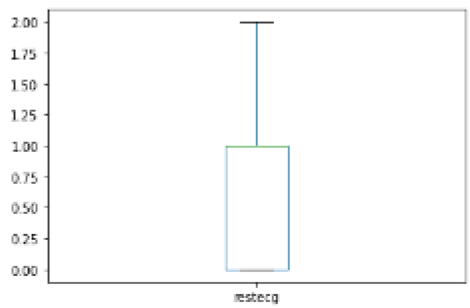

```
[ ] plot_boxplot(data,'chol')
```



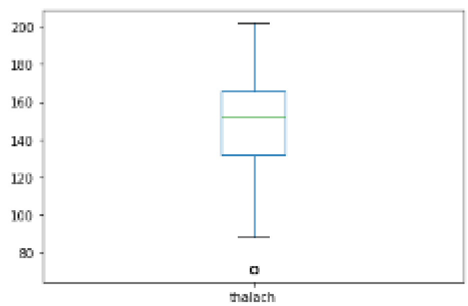
```
[ ] plot_boxplot(data,'fbs')
```



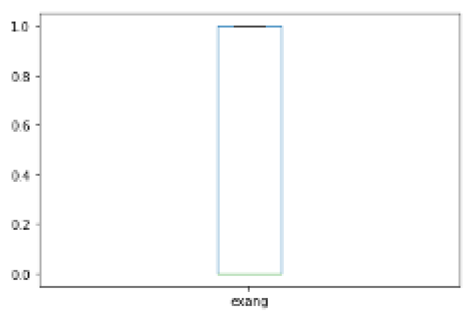

```
[ ] plot_boxplot(data,'restecg')
```



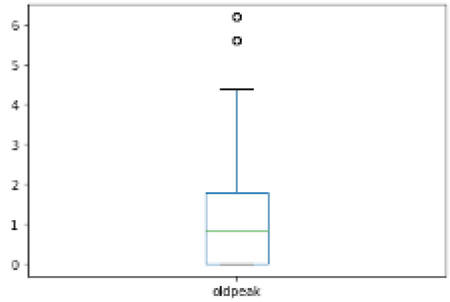

```
[ ] plot_boxplot(data,'thalach')
```



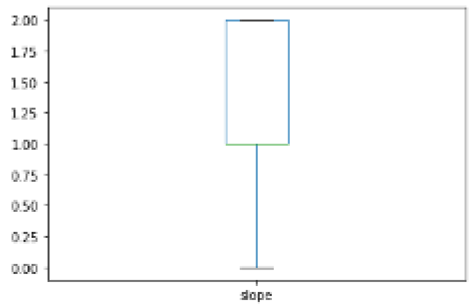

```
[ ] plot_boxplot(data,'exang')
```



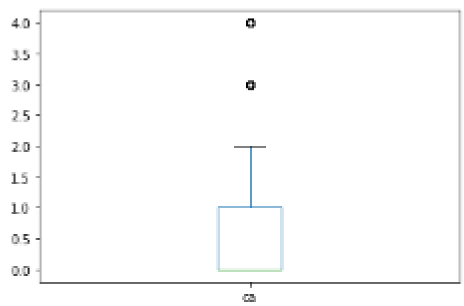
```
[ ] plot_boxplot(data,'oldpeak')
```



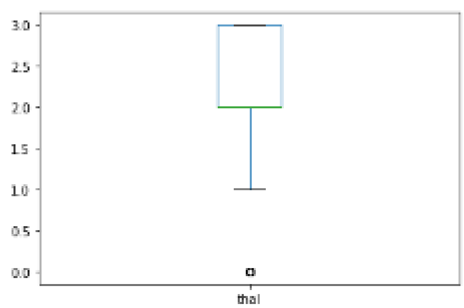

```
[ ] plot_boxplot(data,'slope')
```



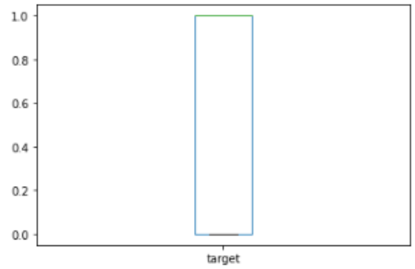

```
[ ] plot_boxplot(data,'ca')
```




```
[ ] plot_boxplot(data,'thal')
```

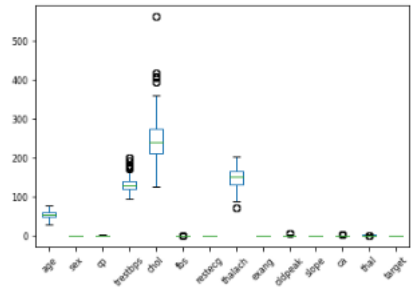


```
[ ] plot_boxplot(data,'target')
```




```
[ ] data.boxplot(grid=False,rot=45,fontsize=8)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fed02a7dc10>




```
[ ] def Outlier(df,ft):
      Q1=df[ft].quantile(0.25)
      Q3=df[ft].quantile(0.75)
      IQR=Q3-Q1
      lower_bound=Q1-1.5*IQR
      upper_bound=Q3+1.5*IQR
      lt=df.index[(df[ft]<lower_bound)|(df[ft]>upper_bound)]
      return lt
```

```
[ ] index_list=[]
    for feature in data.columns:
      index_list.extend(Outlier(data,feature))
```

```
[ ] index_list
```

```
508,
509,
528,
609,
624,
636,
679,
688,
837,
891,
896,
944,
971,
986,
```

```
def remove(df,lt):
  lt=sorted(set(lt))
  df=df.drop(lt)
  return df
```

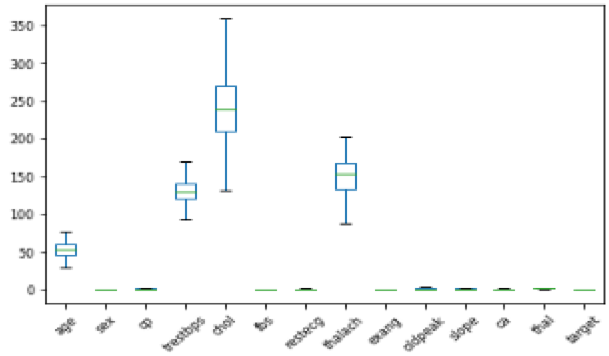
```
data=remove(data,index_list)
```

```
data.shape
```

```
(769, 14)
```

```
data.boxplot(grid=False,rot=45,fontsize=8)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fed02b26390>
```



HANDLING MISSING VALUES

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

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

```
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
5	58	0	0	100	248	0	0	122	0	1.0	1	0	2	1
7	55	1	0	160	289	0	0	145	1	0.8	1	1	3	0

Data Balnacing using Synthetic Minority Over-sampling Technique(SMOTE)

```
[ ] data.target.value_counts()
```

```
1    422
0    347
Name: target, dtype: int64
```

```
[ ] x=data.drop(['target'],axis=1)
    y=data.target
    x.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3
5	58	0	0	100	248	0	0	122	0	1.0	1	0	2
7	55	1	0	160	289	0	0	145	1	0.8	1	1	3

Double-click (or enter) to edit

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.2)
    print(x_train.shape, y_train.shape)
    print(x_test.shape, y_test.shape)
```

```
(615, 13) (615,)
(154, 13) (154,)
```

```
[ ] from sklearn.neighbors import KNeighborsClassifier
```

```
[ ] model=KNeighborsClassifier()
```

```
[ ] model.fit(x_train,y_train)
    y_predict=model.predict(x_test)
```

```
[ ] pip install imblearn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: imblearn in /usr/local/lib/python3.7/dist-packages (0.0)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.7/dist-packages (from imblearn) (0.8.1)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages (from imbalanced-learn->imblearn) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from imbalanced-learn->imblearn) (1.1.0)
Requirement already satisfied: scikit-learn>=0.24 in /usr/local/lib/python3.7/dist-packages (from imbalanced-learn->imblearn) (1.0.2)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from imbalanced-learn->imblearn) (1.21.6)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.24->imbalanced-learn->imblearn) (3.1.0)
```

```
[ ] from sklearn.metrics import accuracy_score
    print(accuracy_score(y_test,y_predict))
    pd.crosstab(y_test,y_predict)
```

0.6623376623376623

col_0	0	1
target		
0	49	25
1	27	53

```
[ ] from imblearn.over_sampling import SMOTE
    smote=SMOTE()
```

```
[ ] x_smote_train,y_smote_train=smote.fit_resample(x_train.astype('float'),y_train)
```

```
[ ] from collections import Counter
    print("Before applying SMOTE:",Counter(y_train))
    print("After applying SMOTE:",Counter(y_smote_train))

    Before applying SMOTE: Counter({1: 342, 0: 273})
    After applying SMOTE: Counter({0: 342, 1: 342})
```

FEATURE SELECTION

```
[ ] from sklearn.feature_selection import SelectKBest
    from sklearn.feature_selection import chi2
```

```
[ ] important_features = SelectKBest(score_func=chi2, k=13)
    fit = important_features.fit(x,y)
    data_scores = pd.DataFrame(fit.scores_)
    data_columns = pd.DataFrame(x.columns)
    features_Scores = pd.concat([data_columns,data_scores],axis=1)
    features_Scores.columns = ['Attributes','Score']
```

```
[ ] featureScores = features_Scores.sort_values(by='Score',ascending=False)
    featureScores
```

	Attributes	Score
7	thalach	451.309044
9	oldpeak	171.425679
11	ca	160.566885
2	cp	120.788355
8	exang	90.361309
0	age	59.672161
4	chol	41.123807
1	sex	32.015546
12	thal	22.046021
10	slope	18.963580
3	trestbps	10.333202
6	restecg	6.108050
5	fbs	NaN

```
[ ] data=data[['thalach','oldpeak','ca','cp','target']]
```

```
[ ] data.head()
```

	thalach	oldpeak	ca	cp	target
0	168	1.0	2	0	0
2	125	2.6	0	0	0
3	161	0.0	1	0	0
5	122	1.0	0	0	1
7	145	0.8	1	0	0

```
[ ] from sklearn.model_selection import train_test_split
```

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

```
[ ] X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

```
[ ] print(X_train.shape,y_train.shape)
    print(X_test.shape,y_test.shape)

    (615, 4) (615,)
    (154, 4) (154,)
```

CHAPTER 6

RESULTS

DATA SET-1: FRAMINGHAM DATASET

```
In [65]:  from sklearn.linear_model import LogisticRegression
          from sklearn import datasets, linear_model
          from imblearn.pipeline import Pipeline
          from sklearn.svm import SVC
          from imblearn.pipeline import Pipeline
          from sklearn.model_selection import RepeatedStratifiedKFold
          from numpy import mean
          from sklearn.datasets import make_classification
          from sklearn.model_selection import cross_val_score
          from sklearn.model_selection import RepeatedStratifiedKFold
          from sklearn.tree import DecisionTreeClassifier
          from imblearn.pipeline import Pipeline
          from imblearn.over_sampling import SMOTE
          from imblearn.under_sampling import RandomUnderSampler
          from imblearn.over_sampling import SMOTE
          import sklearn.linear_model as lm



          lm = lm.LogisticRegression()
          model = lm.fit(X_train, y_train)
          over = SMOTE(sampling_strategy=0.1)
          steps = [('over', over), ('model', model)]
          pipeline = Pipeline(steps=steps)
          X, y = make_classification(n_samples=10000, n_features=5, n_redundant=0,
              n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
          cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
          scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
          print('Mean ROC AUC: %.3f' % mean(scores))

          Mean ROC AUC: 0.943
```

Fig2: Logistic Regression of Framingham Dataset

```
In [66]:  from sklearn.tree import DecisionTreeClassifier
          classifier = DecisionTreeClassifier()
          classifier.fit(X_train, y_train)
          y_pred = classifier.predict(X_test)
          from sklearn.metrics import classification_report, confusion_matrix
          #print(confusion_matrix(y_test, y_pred))
          from sklearn.metrics import accuracy_score
          print(accuracy_score(y_test, y_pred, normalize=True)
          )

          0.7358490566037735

In [68]:  from sklearn.neighbors import KNeighborsClassifier
          classifier = KNeighborsClassifier(n_neighbors=5)
          classifier.fit(X_train, y_train)
          y_pred = classifier.predict(X_test)
          from sklearn.metrics import accuracy_score
          print(accuracy_score(y_test, y_pred, normalize=True)
          )

          0.8254716981132075
```

Fig3: KNN of Framingham Dataset

```
In [67]:  import pandas
          from sklearn import model_selection
          from sklearn.linear_model import LogisticRegression
          from sklearn.tree import DecisionTreeClassifier
          from sklearn.svm import SVC
          from sklearn.ensemble import VotingClassifier

          kfold = model_selection.KFold(n_splits=5)
          estimators = []
          model1 = LogisticRegression().fit(x_test,y_test)
          estimators.append(('logistic', model1))
          model2 = KNeighborsClassifier(n_neighbors=3)
          estimators.append(('cart', model2))

          ensemble = VotingClassifier(estimators)

          over = SMOTE(sampling_strategy=0.1)
          steps = [('over', over), ('model', ensemble)]
          pipeline = Pipeline(steps=steps)


          results = model_selection.cross_val_score(pipeline, X, y, cv=kfold)
          print(results.mean())

          C:\Users\USER\AppData\Roaming\Python\Python38\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs
          failed to converge (status=1):
          STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

          Increase the number of iterations (max_iter) or scale the data as shown in:
              https://scikit-learn.org/stable/modules/preprocessing.html
          Please also refer to the documentation for alternative solver options:
              https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
            n_iter_i = _check_optimize_result(

          0.9917999999999999
```

Fig4: ensemble of KNN and logistic Regression of Framingham Dataset

DATA SET-2: CLEVELAND DATASET

```
# fit a model
lm = lm.LogisticRegression()
model = lm.fit(x_train, y_train)
over = SMOTE(sampling_strategy=0.1)
steps = [('over', over), ('model', model)]
pipeline = Pipeline(steps=steps)
X, y = make_classification(n_samples=10000, n_features=5, n_redundant=0,
n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
#model.score(X_test, y_test)
print('Mean ROC AUC: %.3f' % mean(scores))

C:\Users\Bhavani Omkarini\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(

Mean ROC AUC: 0.943
```

Fig5: Logistic Regression of Cleveland Dataset

```
In [52]: from sklearn.tree import DecisionTreeClassifier
         classifier = DecisionTreeClassifier()
         classifier.fit(x_train, y_train)
         y_pred = classifier.predict(x_test)
         from sklearn.metrics import classification_report, confusion_matrix
         #print(confusion_matrix(y_test, y_pred))
         from sklearn.metrics import accuracy_score
         print(accuracy_score(y_test, y_pred, normalize=True)
         )

         0.7674418604651163

In [53]: from sklearn.neighbors import KNeighborsClassifier
         classifier = KNeighborsClassifier(n_neighbors=5)
         classifier.fit(x_train, y_train)
         y_pred = classifier.predict(x_test)
         from sklearn.metrics import accuracy_score
         print(accuracy_score(y_test, y_pred, normalize=True)
         )

         0.6511627906976745
```

Fig 6: KNN and Logistic Regression of Cleveland Dataset

```
In [48]: import pandas
         from sklearn import model_selection
         from sklearn.linear_model import LogisticRegression
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.svm import SVC
         from sklearn.ensemble import VotingClassifier

         kfold = model_selection.KFold(n_splits = 5)

         #create the sub models
         estimators = []
         model1 = LogisticRegression().fit(x_test, y_test)
         estimators.append(('logistic', model1))
         model2 = KNeighborsClassifier(n_neighbors = 3)
         estimators.append(('cart',model2))

         #create the ensemble model
         ensemble = VotingClassifier(estimators)

         over = SMOTE(sampling_strategy = 0.1)
         steps = [('over', over), ('model',ensemble)]
         pipeline = Pipeline(steps = steps)

         results = model_selection.cross_val_score(pipeline, x, y, cv = kfold)
         print(results.mean())

C:\Users\Bhavani Omkarini\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(

0.9915
```

Fig 7: ensemble of KNN and logistic Regression of Cleveland Dataset

DATA SET-3: HEART DISEASE DATASET

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn import datasets, linear_model
    from imblearn.pipeline import Pipeline
    from sklearn.svm import SVC
    from imblearn.pipeline import Pipeline
    from sklearn.model_selection import RepeatedStratifiedKFold
    # decision tree  on imbalanced dataset with SMOTE oversampling and random undersampling
    from numpy import mean
    from sklearn.datasets import make_classification
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import RepeatedStratifiedKFold
    from sklearn.tree import DecisionTreeClassifier
    from imblearn.pipeline import Pipeline
    from imblearn.over_sampling import SMOTE
    from imblearn.under_sampling import RandomUnderSampler
    from imblearn.over_sampling import SMOTE
    import sklearn.linear_model as lm



    # fit a model
    lm = lm.LogisticRegression()
    model = lm.fit(X_train, y_train)
    over = SMOTE(sampling_strategy=0.1)
    steps = [('over', over), ('model', model)]
    pipeline = Pipeline(steps=steps)
    X, y = make_classification(n_samples=10000, n_features=5, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
    scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
    #model.score(X_test, y_test)
    print('Mean ROC AUC: %.3f' % mean(scores))

    Mean ROC AUC: 0.943
```

Fig 8: Logistic Regression of Heart Disease Dataset

```
[55] from sklearn.neighbors import KNeighborsClassifier
     classifier = KNeighborsClassifier(n_neighbors=5)
     classifier.fit(X_train, y_train)
     y_pred = classifier.predict(X_test)
     from sklearn.metrics import accuracy_score
     print(accuracy_score(y_test, y_pred, normalize=True)
     )

     0.8181818181818182
```

Fig 9: KNN of Heart Disease Dataset

```
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

kfold = model_selection.KFold(n_splits=5)
# create the sub models
estimators = []
model1 = LogisticRegression().fit(x_test,y_test)
estimators.append(('logistic', model1))
model2 = KNeighborsClassifier(n_neighbors=3)
estimators.append(('cart', model2))

# create the ensemble model
ensemble = VotingClassifier(estimators)

over = SMOTE(sampling_strategy=0.1)
steps = [('over', over), ('model', ensemble)]
pipeline = Pipeline(steps=steps)


results = model_selection.cross_val_score(pipeline, X, y, cv=kfold)
print(results.mean())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
0.9921
```

Fig 10: Ensemble of KNN and Logistic Regression in Heart Disease Dataset

CHAPTER 7

CONCLUSION

Our framework initially did Data Pre-processing. In Data Pre-processing, Outlier and Null values were removed and Data Balancing was done. After Data Pre-processing, Feature Selection was done by calculating p-score. Lastly an Ensemble of Logistic Regression and KNN was proposed.

Data set used is Framingham Dataset. For validation, other two data sets were used Heart-Disease and Cleveland. Before Applying Ensemble Algorithm, we tried applying only Logistic and KNN algorithm separately. After applying Logistic Algorithm to Framingham Dataset, the accuracy that we got was 94%. Similarly, after applying KNN Algorithm to Framingham Dataset, 82% accuracy was achieved. But after applying Ensemble of KNN and Logistic Regression, the accuracy is 99%. Similarly, we did this for other two datasets Heart-Disease (Logistic Regression: 94%, KNN: 81%, Ensemble: 99%) and Cleveland (Logistic Regression: 94%, KNN: 65%, Ensemble: 99.15%) for validation. From these results, we can conclude that our framework gives high accuracy.

FUTURE WORKS

In future, More than two Ensemble algorithm can be used to get more accuracy

CHAPTER 8

REFERENCES

- S. Ambekar and R. Phalnikar, ‘‘Disease risk prediction by using convolutional neural network,’’ in Proc. 4th Int. Conf. Comput. Commun. Control Autom. (ICCUBEA), Aug. 2018, pp. 1–5.
- Health Stats 2017 by World Health Organization (WHO). Accessed: Mar. 2021. [Online]. Available: https://www.who.int/news-room/factsheets/detail/cardiovascular-diseases-(CVDs)
- D. Mousa, N. Zayed, and I. A. Yassine, ‘‘Automatic cardiac MRI localization method,’’ in Proc. Cairo Int. Biomed. Eng. Conf. (CIBEC), Giza, Egypt, Dec. 2014, pp. 153–157.
- L.-N. Pu, Z. Zhao, and Y.-T. Zhang, ‘‘Investigation on cardiovascular risk prediction using genetic information,’’ IEEE Trans. Inf. Technol. Biomed., vol. 16, no. 5, pp. 795–808, Sep. 2012.
- V. Gil-Guillen, D. Orozco-Beltran, A. Maiques-Galan, J. Aznar-Vicente, J. Navarro, L. Cea-Calvo, F. Quirze-Andrés, J. Redon, and J. Merino-Sanchez, ‘‘Agreement between REGICOR and SCORE scales in identifying high cardiovascular risk in the Spanish population,’’ Revista Espanola de Cardiol., vol. 60, no. 10, p. 1042, 2007.