A TPCS REPORT ON

PREDICTION OF PREDOMINANT DISEASES IN TRIBAL GADCHIROLI USING MACHINE LEARNING ALGORITHMS

Submitted by

Shivam Kadukar

Under the guidance of

Prof. Ramesh Kate

DR. BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY,LONERE

ACADEMIC YEAR 2020-21

DR. BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY,LONERE

DEPARTMENT OF MECHANICAL ENGINEERING

CERTIFICATE

This is to certify that the following student of Third Year B-Tech Programme in Mechanical Engineering from Dr. Babasaheb Ambedkar Technological University, Lonere have successfully completed the project work titled, **"Prediction of Predominant Disease in Tribal Gadchiroli using Machine Learning Algorithms"** during the academic year 2020-21, under the guidance of Prof. Ramesh Kate.

Name of Students	PRN.
Shivam Sanjay Kadukar	10303320181161210006

Head of	Guide
Mechanical Engg. Dept.	Prof. Ramesh Kate
Prof. Neeraj Agrawal	

External Examiner

Date: -

Place:- Dr. Babasaheb Ambedkar Technological University, Lonere

UNDERTAKING

I declare that the work presented in this report titled, "**Prediction of Predominant Diseases in Tribal Gadchiroli using Machine Learning Algorithms"**, submitted to the **Mechanical Engineering Department, Dr. Babasaheb Ambedkar Technological University, Lonere** , to fulfil the requirements for the TPCS in 3rd year of B.Tech mechanical engineering, and is my original work. We have not plagiarized or submitted the same work for the award of any other degree or examination. In case this undertaking found incorrect, we accept that our B-Tech may be unconditionally withdrawn or may be punishable as per the norms of the institute.

Name of the Students	PRN.	SIGN
Shivam Sanjay Kadukar	10303320181161210006	

Date:-
Place:- Dr. Babasaheb Ambedkar Technological University, Lonere

ACKNOWLEDGEMENT

I feel great pleasure in submitting this project report on "**Prediction of Predominant Diseases in Tribal Gadchiroli using Machine Learning Algorithms"**, we wish to express true sense of gratitude towards our project guide **PROF. RAMESH KATE** who at very discrete step in of this project contributed with valuable guidance every problem that arose.

We wish to, ones again thanks to our **PROF. NEERAJ AGRAWAL**, Head of Department, Mechanical Engineering Department for opening the doors of department towards the realization of project report.

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

Medical artificial intelligence (AI) mainly uses computer techniques to perform clinical diagnoses and suggest treatments. AI has the capability of detecting meaningful relationships in a data set and has been widely used in many clinical situations to diagnose, treat, and predict the results.
Artificial intelligence in medicine and healthcare has been a particularly hot topic in recent years. There is a sense of great potential in the application of AI in medicine. AI can be used to list possible diagnosis on basis on symptoms, to designed to detect, track and investigate infections in hospitalized patients, as AI therapy and to reduce human errors while surgeries.

Gadchiroli is a district in State of Maharashtra with majority of tribal population. This is state shares its border with State of Chattisgarh and Telangana and hence comes under the Red Corridor(Laxal Affected Belts).
The district is among the underdeveloped district of India. “The villagers have to travel 200 km to Nagpur to access a fully-equipped healthcare facility,” Dr. Pankaj Chaturvedi, said deputy director of Tata Memorial Centre in a study done in 2015- 16. Many attempts of development in various fields including medical and health are done and the conditions are good than it was before few years but still there is lot of work to be done.

This project does an attempt to introduce Artificial Intelligence in medical field of Gadchiroli district by shortlisting the predominant diseases in the district and building machine learning algorithms from prediction models.

CHAPTER 2: OBJECTIVES

- To find out the diseases which are predominant in the district of Gadchiroli.
- To find out the reason of large number of cases of this particular diseases.
- To build various machine learning models to perform prediction of this diseases.
- To check with of this models works best and has higher accuracy in prediction of the diseases.

CHAPTER 3: METHODOLOGY

- Various government documents and News Article reporting the diseases in Gadchiroli were checked to short list the most predominant diseases and the diseases which can be fatal if the diagnosis is late or ignored.

- An attempt to find reason why the large number of cases of this diseases are seen in tribal Gadchiroli district was made by researching over this documents and news articles.

- The symptoms of diseases, the test that are done to predict the report of a patients were noted. Also, what features are measured to provide the result as positive or negative were noted.

- Then, on the basis of the above findings patients report data was collected for each disease over the internet.

- With the help of this collected data, Machine Learning models were build using various algorithms to predict the report of the patients.

- The performance of these models were validated.

- These models were compared on the basis of various performance measurements to infer the best model which predict the result most precisely.

CHAPTER 4: PREDOMINANT DISEASES IN GADCHIROLI

"**Gadchiroli in Maharashtra records highest rate of oral cancer in India**", Hindustan Times reported in a news article in April 2019.

A joint study was conducted by Tata Memorial Hospital (TMH) and Dr Abhay Bang's Society for Education, Action and Research in Community Health (SEARCH). It found that the incidence rate of oral cancer in Gadchiroli was **12 cases per one lakh population in women and 20 cases per one lakh in men in the year 2015-16**, which was the highest in the country. The district reports over 600 cases of cancer every year with 40% of all cancers in men and 20% among women identified as oral cancers. Nationally, the prevalence of oral cancer is about 10-12% of all cancer cases.

"Mouth was the leading cancer site among men, and the second leading site among women after the cervix and breast," the study said. The cervix and breast are the leading cancer site among women in Gadchiroli District.

A report published by 'Down To Earth Organisation' published in September 2018 stated that," tribal people or scheduled tribes, who constitute 8.6 per cent of India's total population, are actually facing triple burden of diseases: communicable diseases (**malaria, tuberculosis, leprosy** etc.), non-communicable diseases (**diabetes, cardiovascular and cancers**) and mental health problems like stress, substance abuse and so on."

Among communicable diseases, malaria continues to be a major health burden in tribal areas. **Though tribal communities are just 8 percent of the population, they account for 30 percent of all malaria cases** and 60 percent of p falciparum cases and 50 percent of total malaria mortality.

n 2005 India officially declared ‘leprosy elimination’ – defined as a national Prevalence Rate (PR) , 1/10 000.1 However, it was known that there remained districts in some states where PRs were higher than that figure.

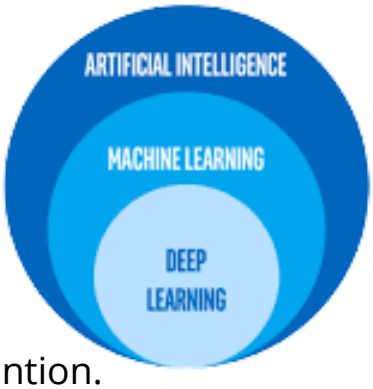
"Highest leprosy rate reported from Gadchiroli, Chandrapur district", a news article by Time of India in December 2018. Gadchiroli has the highest incidence of 5.05 per 10,000 population followed by Chandrapur were this figure 4.04.

Malnutrition and anemia form major public health problems among the school age children, particularly in the developing countries. A study 'Community based screening and management of adolescent anemia in tribal areas of India key to reduction in maternal mortality' was conducted in 2011 by Joshi A. It was found found **95.3% prevalence of anemia in a group of tribal women.**

Nearly 40% of all cancer cases in the underdeveloped district of Gadchiroli are tobacco-related.
"The main addiction in the region is kharra, a mixture of betel nut and scented tobacco, nus, a form of powdered tobacco that women use on their teeth and gudakhu, a paste of tobacco commonly used by men. One can find shops selling tobacco every few kilometres,” said Dr. Yogeshwar Kalkonde, a co-investigator of the study conducted by TMH and SEARCH.

CHAPTER 5:
GLOSSARY : MACHINE LEARNING

NOTE - The glossary is not in alphabetical order but in the order we encounter the terms while building the model for better understanding.



Machine Learning - Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

Python - Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation.

Dataset - A data set (or dataset) is a collection of data.

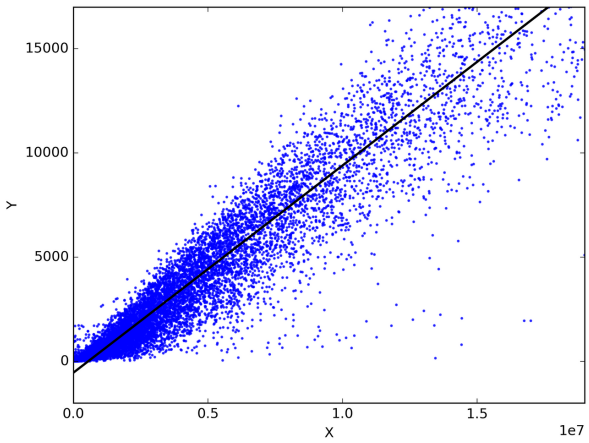
Numpy - NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

Pandas - Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Matplotlib - Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

Scikit-learn(Sklearn) - Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines.

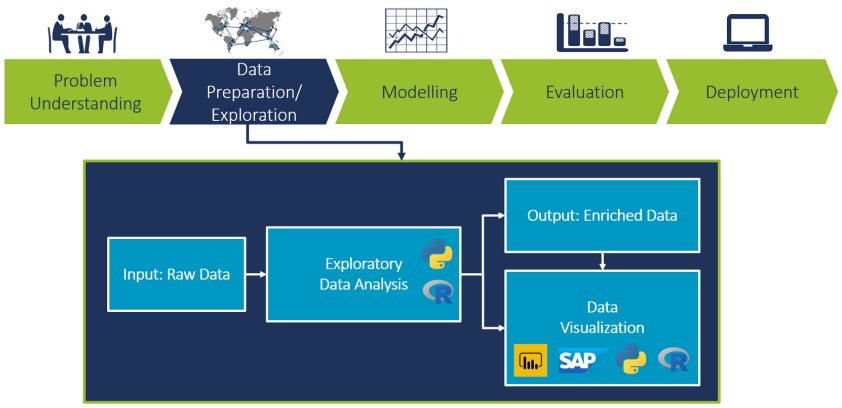
Regression - Regression is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between one dependent variable (usually denoted by Y) and a series of other v ariables (known as independent variables).



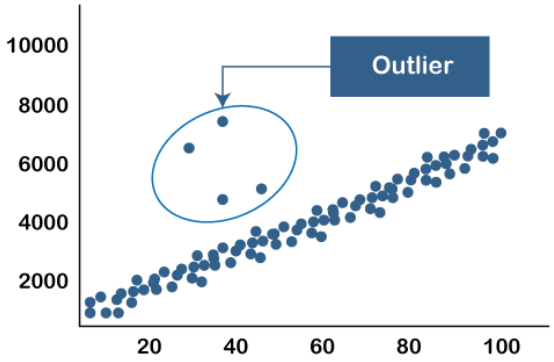
Classification Models - A classification model tries to draw some conclusion from the input values given for training. It will predict the class labels/categories for the new data.

.csv files - A comma-separated values file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas.

Exploratary data analysis - In statistics, exploratory data analysis is an approach of analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.



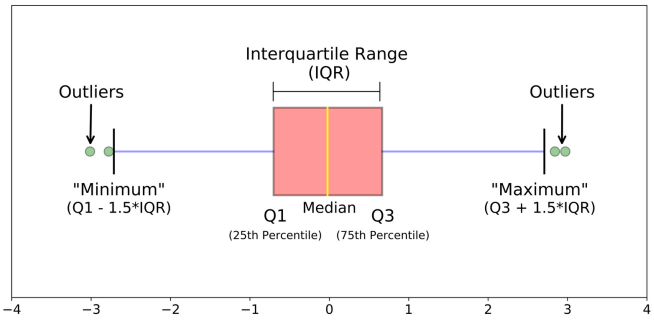
Outliers - In statistics, an outlier is a data point that differs significantly from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the data set.



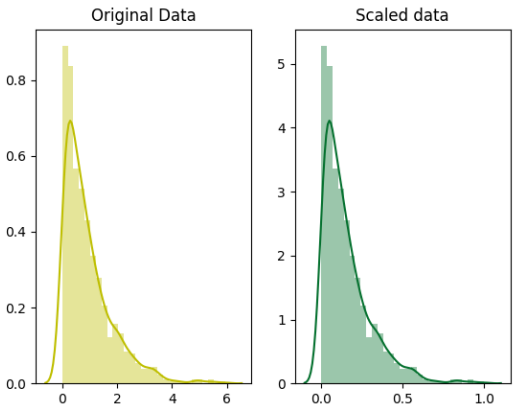
An outlier can cause serious problems in statistical analysis.

BoxPlot - A box plot or boxplot is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending from the boxes indicating variability outside the upper and lower quartiles, hence the terms box-and-whisker plot and box-and-whisker diagram.

Interquartile Range(IQR) - A measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles, IQR = $Q_3 - Q_1$.



Feature Scaling - Feature scaling (also known as data normalization) is the method used to standardize the range of features of data. Since, the range of values of data may vary widely, it becomes a necessary step in data preprocessing while using machine learning algorithms.



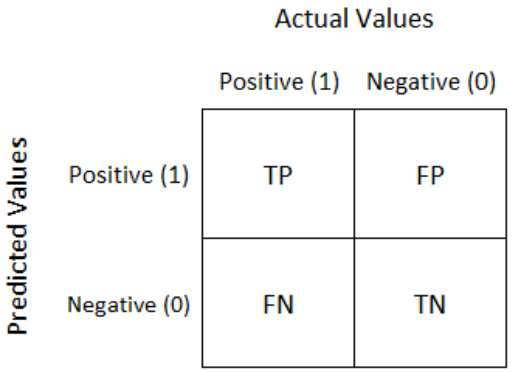
In scaling (also called min-max scaling), you transform the data such that the features are within a specific range e.g. [0, 1].

Multicollinearity- Multicollinearity is the occurrence of high intercorrelations among two or more independent variables in a multiple regression model. In general, multicollinearity can lead to wider confidence intervals that produce less reliable probabilities in terms of the effect of independent variables in a model.

Train-Test Evaluation - The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- Train Dataset: Used to fit the machine learning model.
- Test Dataset: Used to evaluate the fit machine learning model.

Confusion Matrix - A confusion matrix, also known as an error matrix,is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class.



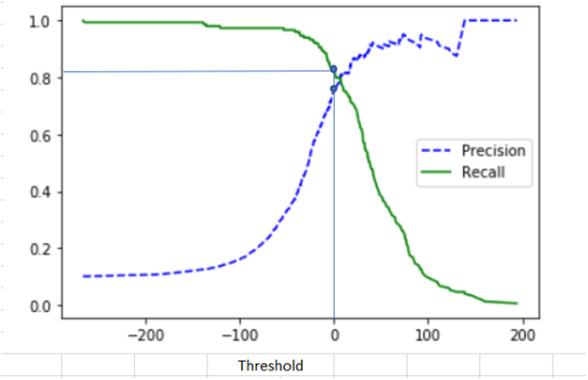
Precision - The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
The best value is 1 and the worst value is 0.

Recall - The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.
The best value is 1 and the worst value is 0.

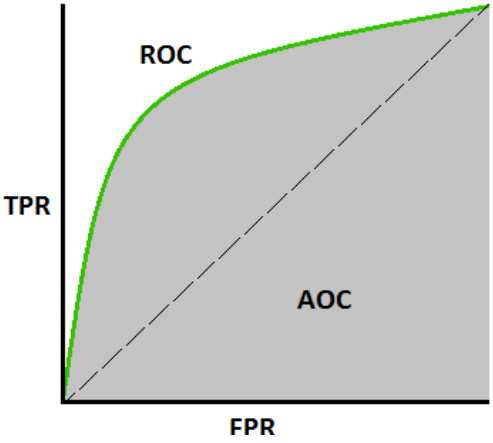
F1 Score - The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

Precision - Recall Tradeoff - The precision-reca curve shows the tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision r elates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).



AUC-ROC CURVE - AUC(area under the curve) - ROC(receiver operating characteristics) curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s.



$$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN} \qquad \text{Specificity} = \frac{TN}{TN + FP}$$

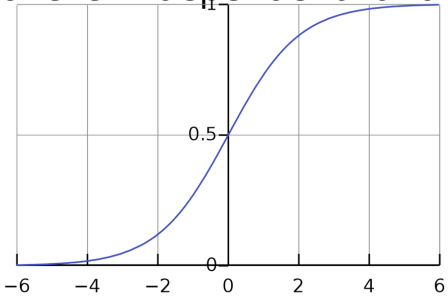
$$\text{FPR} = 1 - \text{Specificity} = \frac{FP}{TN + FP}$$

CLASSIFICATION MODELS -

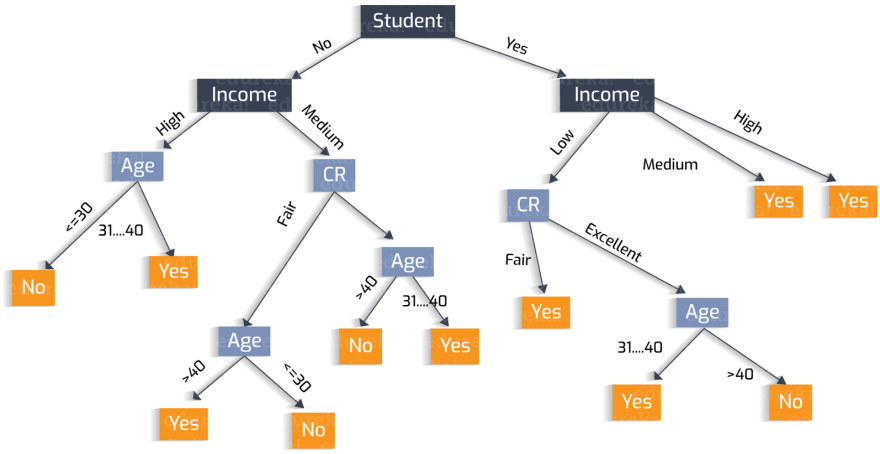
Logistic Regression - Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

- Sigmoid Function -

$$S(x) = \frac{1}{1 + e^{-x}}$$

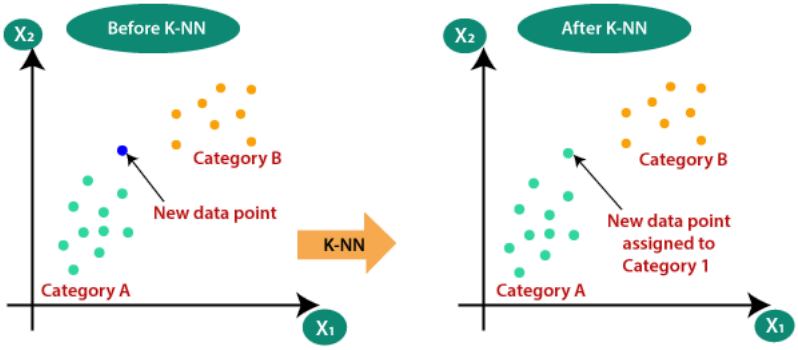


Decision Tree - Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute. This process is then repeated for the subtree rooted at the new node.

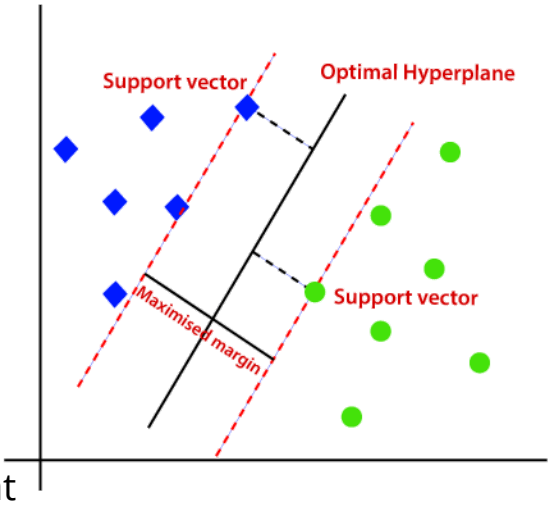


K-nearest neighbours - K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.



Support vector Machine - Support Vector Machine(SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.



CHAPTER 6:
MACHINE LEARNING MODELS

CERVICAL CANCER PREDICTION

Cervical Cancer Prediction

1) Importing the necessary python libraries

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

2)Importing the data set

The data set was stored as .csv files

In [2]:
```
data = pd.read_csv('cervical_cancer.csv')
data.head(10)
```

Out[2]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormo Contracepti (ye
0	18	4.0	15.0	1.0	0.0	0.0	0.0	0.0	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	0.0	
2	34	1.0	?	1.0	0.0	0.0	0.0	0.0	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0	
5	42	3.0	23.0	2.0	0.0	0.0	0.0	0.0	
6	51	3.0	17.0	6.0	1.0	34.0	3.4	0.0	
7	26	1.0	26.0	3.0	0.0	0.0	0.0	1.0	
8	45	1.0	20.0	5.0	0.0	0.0	0.0	0.0	
9	44	3.0	15.0	?	1.0	1.266972909	2.8	0.0	

10 rows × 36 columns

Exploratory Data Analysis

3) Data cleaning

Sometimes the dataset contains error in entries and hence imputation of this entries is required to proceed the model building.

On initial look at the dataset, we can see that some columns contains '?' entries which are of no use hence they are imputed as 'NaN'(not a number) and considered as a missing value in that cells.

In [3]:
```
data.replace(to_replace = '?',value = np.nan, inplace = True)
data.isnull().sum()
```

Out[3]:
```
Age                                   0
Number of sexual partners            26
First sexual intercourse              7
Num of pregnancies                   56
Smokes                               13
```

```
Smokes (years)                       13
Smokes (packs/year)                  13
Hormonal Contraceptives             108
Hormonal Contraceptives (years)     108
IUD                                 117
IUD (years)                         117
STDs                                105
STDs (number)                       105
STDs:condylomatosis                 105
STDs:cervical condylomatosis        105
STDs:vaginal condylomatosis         105
STDs:vulvo-perineal condylomatosis  105
STDs:syphilis                       105
STDs:pelvic inflammatory disease    105
STDs:genital herpes                 105
STDs:molluscum contagiosum          105
STDs:AIDS                           105
STDs:HIV                            105
STDs:Hepatitis B                    105
STDs:HPV                            105
STDs: Number of diagnosis             0
STDs: Time since first diagnosis    787
STDs: Time since last diagnosis     787
Dx:Cancer                             0
Dx:CIN                                0
Dx:HPV                                0
Dx                                    0
Hinselmann                            0
Schiller                              0
Citology                              0
Biopsy                                0
dtype: int64
```

In [4]:

```
data.head(10)
```

Out[4]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormo Contracepti (ye
0	18	4.0	15.0	1.0	0.0	0.0	0.0	0.0	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	0.0	
2	34	1.0	NaN	1.0	0.0	0.0	0.0	0.0	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0	
5	42	3.0	23.0	2.0	0.0	0.0	0.0	0.0	
6	51	3.0	17.0	6.0	1.0	34.0	3.4	0.0	
7	26	1.0	26.0	3.0	0.0	0.0	0.0	1.0	
8	45	1.0	20.0	5.0	0.0	0.0	0.0	0.0	
9	44	3.0	15.0	NaN	1.0	1.266972909	2.8	0.0	

10 rows × 36 columns

The dataset contains total 858 entries and the reports are created on the basis of 36 different features (36 columns). As the '?' entries are replaced with 'NaN' the dataset contains missing values. The number of missing values in each cell can be seen below.

In [5]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 858 entries, 0 to 857
Data columns (total 36 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
```

```
 0   Age                                 858 non-null    int64
 1   Number of sexual partners           832 non-null    object
 2   First sexual intercourse            851 non-null    object
 3   Num of pregnancies                  802 non-null    object
 4   Smokes                              845 non-null    object
 5   Smokes (years)                      845 non-null    object
 6   Smokes (packs/year)                 845 non-null    object
 7   Hormonal Contraceptives             750 non-null    object
 8   Hormonal Contraceptives (years)     750 non-null    object
 9   IUD                                 741 non-null    object
 10  IUD (years)                         741 non-null    object
 11  STDs                                753 non-null    object
 12  STDs (number)                       753 non-null    object
 13  STDs:condylomatosis                 753 non-null    object
 14  STDs:cervical condylomatosis        753 non-null    object
 15  STDs:vaginal condylomatosis         753 non-null    object
 16  STDs:vulvo-perineal condylomatosis  753 non-null    object
 17  STDs:syphilis                       753 non-null    object
 18  STDs:pelvic inflammatory disease    753 non-null    object
 19  STDs:genital herpes                 753 non-null    object
 20  STDs:molluscum contagiosum          753 non-null    object
 21  STDs:AIDS                           753 non-null    object
 22  STDs:HIV                            753 non-null    object
 23  STDs:Hepatitis B                    753 non-null    object
 24  STDs:HPV                            753 non-null    object
 25  STDs: Number of diagnosis           858 non-null    int64
 26  STDs: Time since first diagnosis    71 non-null     object
 27  STDs: Time since last diagnosis     71 non-null     object
 28  Dx:Cancer                           858 non-null    int64
 29  Dx:CIN                              858 non-null    int64
 30  Dx:HPV                              858 non-null    int64
 31  Dx                                  858 non-null    int64
 32  Hinselmann                          858 non-null    int64
 33  Schiller                            858 non-null    int64
 34  Citology                            858 non-null    int64
 35  Biopsy                              858 non-null    int64
dtypes: int64(10), object(26)
memory usage: 241.4+ KB
```

Some of the columns such as 'STDs: Time since first diagnosis' and 'STDs: Time since last diagnosis contains many missing values. In this columns out of 858 total entries only 71 entries has a valid value Imputing so many missing values make the model biased.

The feature column 'Biopsy' is our target variable. and all the other are 'independent variables'

Hence, this columns are removed from the dataset.

In [6]:
```
data.drop(['STDs: Time since first diagnosis','STDs: Time since last
diagnosis'],
          axis = 'columns', inplace = True)
```

Below we can see all the columns(features) are present in the dataset.

In [7]:
```
data.columns
```

Out[7]:
```
Index(['Age', 'Number of sexual partners', 'First sexual intercourse',
       'Num of pregnancies', 'Smokes', 'Smokes (years)', 'Smokes (packs/year)',
       'Hormonal Contraceptives', 'Hormonal Contraceptives (years)', 'IUD',
       'IUD (years)', 'STDs', 'STDs (number)', 'STDs:condylomatosis',
       'STDs:cervical condylomatosis', 'STDs:vaginal condylomatosis',
       'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis',
       'STDs:pelvic inflammatory disease', 'STDs:genital herpes',
       'STDs:molluscum contagiosum', 'STDs:AIDS', 'STDs:HIV',
       'STDs:Hepatitis B', 'STDs:HPV', 'STDs: Number of diagnosis',
       'Dx:Cancer', 'Dx:CIN', 'Dx:HPV', 'Dx', 'Hinselmann', 'Schiller',
       'Citology', 'Biopsy'],
      dtype='object')
```

Duplicate entries in the dataset are of no use and can also make the model to overfit and give bad

performance hence, they are removed. After removal of missing columns and duplicate entries the dataset now has 835 entries and 35 feature columns.

In [8]:
```
data = data.drop_duplicates()
```

In [9]:
```
data.shape
```

Out[9]:
```
(835, 34)
```

In [10]:
```
data_numerical = data.select_dtypes(include = 'number')
data_numerical.dtypes
```

Out[10]:
```
Age                                 int64
STDs: Number of diagnosis           int64
Dx:Cancer                           int64
Dx:CIN                              int64
Dx:HPV                              int64
Dx                                  int64
Hinselmann                          int64
Schiller                            int64
Citology                            int64
Biopsy                              int64
dtype: object
```

In [11]:
```
data_categorical = data.select_dtypes(include = 'object')
data_categorical1 = data_categorical.apply(pd.to_numeric)
data_categorical1.dtypes
```

Out[11]:
```
Number of sexual partners           float64
First sexual intercourse            float64
Num of pregnancies                  float64
Smokes                              float64
Smokes (years)                      float64
Smokes (packs/year)                 float64
Hormonal Contraceptives             float64
Hormonal Contraceptives (years)     float64
IUD                                 float64
IUD (years)                         float64
STDs                                float64
STDs (number)                       float64
STDs:condylomatosis                 float64
STDs:cervical condylomatosis        float64
STDs:vaginal condylomatosis         float64
STDs:vulvo-perineal condylomatosis  float64
STDs:syphilis                       float64
STDs:pelvic inflammatory disease    float64
STDs:genital herpes                 float64
STDs:molluscum contagiosum          float64
STDs:AIDS                           float64
STDs:HIV                            float64
STDs:Hepatitis B                    float64
STDs:HPV                            float64
dtype: object
```

In [12]:
```
data = pd.concat([data_numerical,data_categorical1],axis = 1,join =
'inner')
data.head(10)
```

Out[12]:

	Age	STDs: Number of diagnosis	Dx:Cancer	Dx:CIN	Dx:HPV	Dx	Hinselmann	Schiller	Citology	Biopsy	...	STDs:v condyloma
0	18	0	0	0	0	0	0	0	0	0	...	

1	15	0	0	0	0	0	0	0	0	0	...
2	34	0	0	0	0	0	0	0	0	0	...
3	52	0	1	0	1	0	0	0	0	0	...
4	46	0	0	0	0	0	0	0	0	0	...
5	42	0	0	0	0	0	0	0	0	0	...
6	51	0	0	0	0	0	1	1	0	1	...
7	26	0	0	0	0	0	0	0	0	0	...
8	45	0	1	0	1	1	0	0	0	0	...
9	44	0	0	0	0	0	0	0	0	0	...

10 rows × 34 columns

4) Imputing the missing values

The missing values are imputed in the following manner: If the feature is of numerical type then the missing values are imputed by 'mean' or 'median'. 'Median' is prefered more often.

If the feature is of categorical type then the missing values are imputed by 'mode' the values are seen most frequent in that column.

Imputation of Target variable is not advisable since it make the model learn from the biases. Removal of entire row is prefered in this case.

In [13]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'median')
data[data_numerical.columns] =
imputer.fit_transform(data[data_numerical.columns])
```

In [14]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy =
'most_frequent')
data[data_categorical1.columns] =
imputer.fit_transform(data[data_categorical1.columns])
```

In [15]:
```
data.isnull().sum()
```

Out[15]:
```
Age                                 0
STDs: Number of diagnosis           0
Dx:Cancer                           0
Dx:CIN                              0
Dx:HPV                              0
Dx                                  0
Hinselmann                          0
Schiller                            0
Citology                            0
Biopsy                              0
Number of sexual partners           0
First sexual intercourse            0
Num of pregnancies                  0
Smokes                              0
Smokes (years)                      0
Smokes (packs/year)                 0
Hormonal Contraceptives             0
Hormonal Contraceptives (years)     0
IUD                                 0
```

```
IUD (years)                         0
STDs                                0
STDs (number)                       0
STDs:condylomatosis                 0
STDs:cervical condylomatosis        0
STDs:vaginal condylomatosis         0
STDs:vulvo-perineal condylomatosis  0
STDs:syphilis                       0
STDs:pelvic inflammatory disease    0
STDs:genital herpes                 0
STDs:molluscum contagiosum          0
STDs:AIDS                           0
STDs:HIV                            0
STDs:Hepatitis B                    0
STDs:HPV                            0
dtype: int64
```

In [16]:
```
data.describe()
```

Out[16]:

	Age	STDs: Number of diagnosis	Dx:Cancer	Dx:CIN	Dx:HPV	Dx	Hinselmann	Schiller	8
count	835.000000	835.000000	835.000000	835.000000	835.000000	835.000000	835.000000	835.000000	
mean	27.023952	0.089820	0.021557	0.010778	0.021557	0.028743	0.041916	0.087425	
std	8.482986	0.306335	0.145319	0.103320	0.145319	0.167182	0.200518	0.282626	
min	13.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	21.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	26.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	32.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
max	84.000000	3.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 34 columns

5) Treating the Outliers

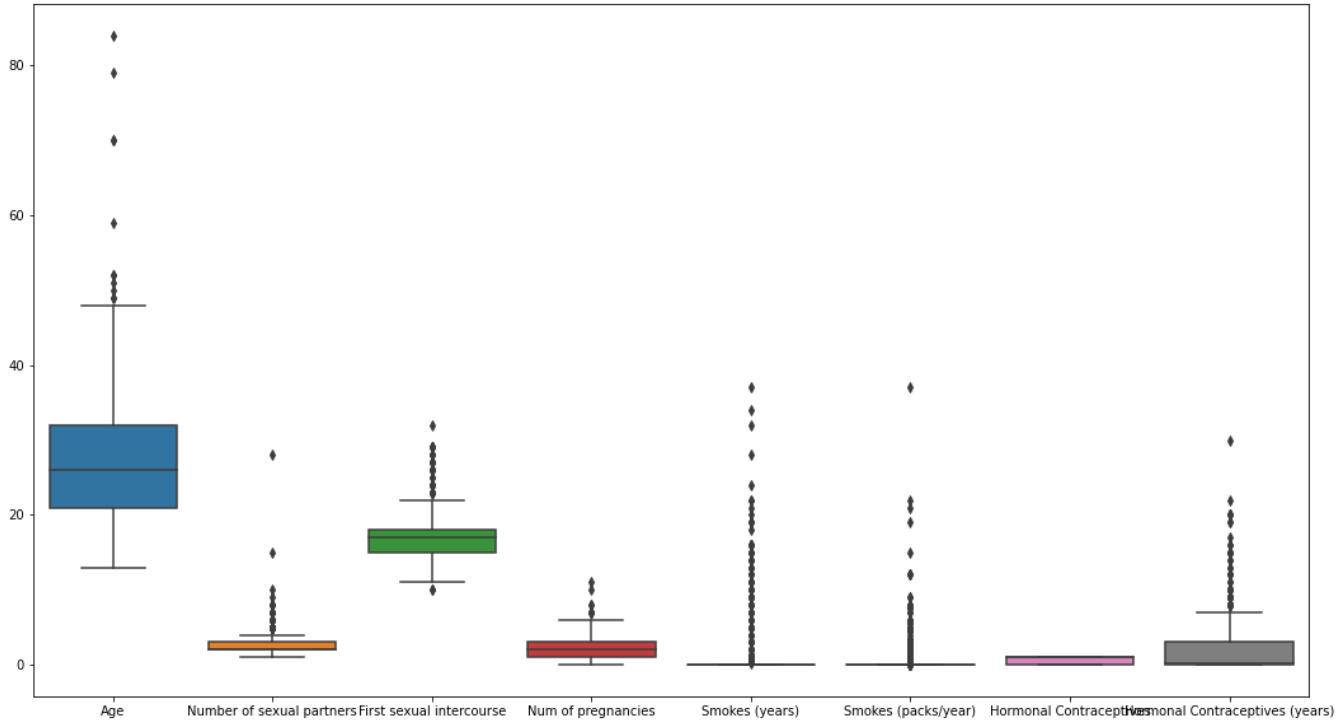
The dataset can also contain outliers in the numerical datatypes columns. This values must be treated since this outliers can affect the predictions performed by the model.

In [17]:
```
numerical_columns=['Age','Number of sexual partners','First sexual
intercourse','Num of pregnancies','Smokes (years)',
 'Smokes (packs/year)','Hormonal Contraceptives','Hormonal
Contraceptives (years)']
```

The boxplots for the numerical columns below shows that it contains outliers and hence they are needed to be treated.

In [18]:
```
plt.figure(figsize = (18,10))
sns.boxplot(data=data[numerical_columns])
```

Out[18]:
```
<AxesSubplot:>
```



If the outlier value is less than the 25th quartile value then it is replaced by lower_limit which is equal to q1 - 1.5*iqr.

If the outlier value is greater than the 75th quartile value then it is replaced by upper_limit which is equal to q3 + 1.5*iqr.

where q1 = 25th quartile, q3 = 75th quartile, iqr = q3 -q1.

In [19]:

```
def cap_data(data):
    for col in numerical_columns:
        if (((data[col].dtype)=='float64') |
((data[col].dtype)=='int64')):
            percentiles = data[col].quantile([0.25,0.75]).values
            iqr = percentiles[1]-percentiles[0]
            upper_limit = percentiles[1] + 1.5*iqr
            lower_limit = percentiles[0] -1.5*iqr
            data[col][data[col] <= percentiles[0]] = lower_limit
            data[col][data[col] >= percentiles[1]] = upper_limit
        else:
            data[col]=data[col]
    return data

cap_data(data)
```

Out[19]:

	Age	STDs: Number of diagnosis	Dx:Cancer	Dx:CIN	Dx:HPV	Dx	Hinselmann	Schiller	Citology	Biopsy	...	STDs condylo
0	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
1	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
2	48.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
3	48.5	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	
4	48.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

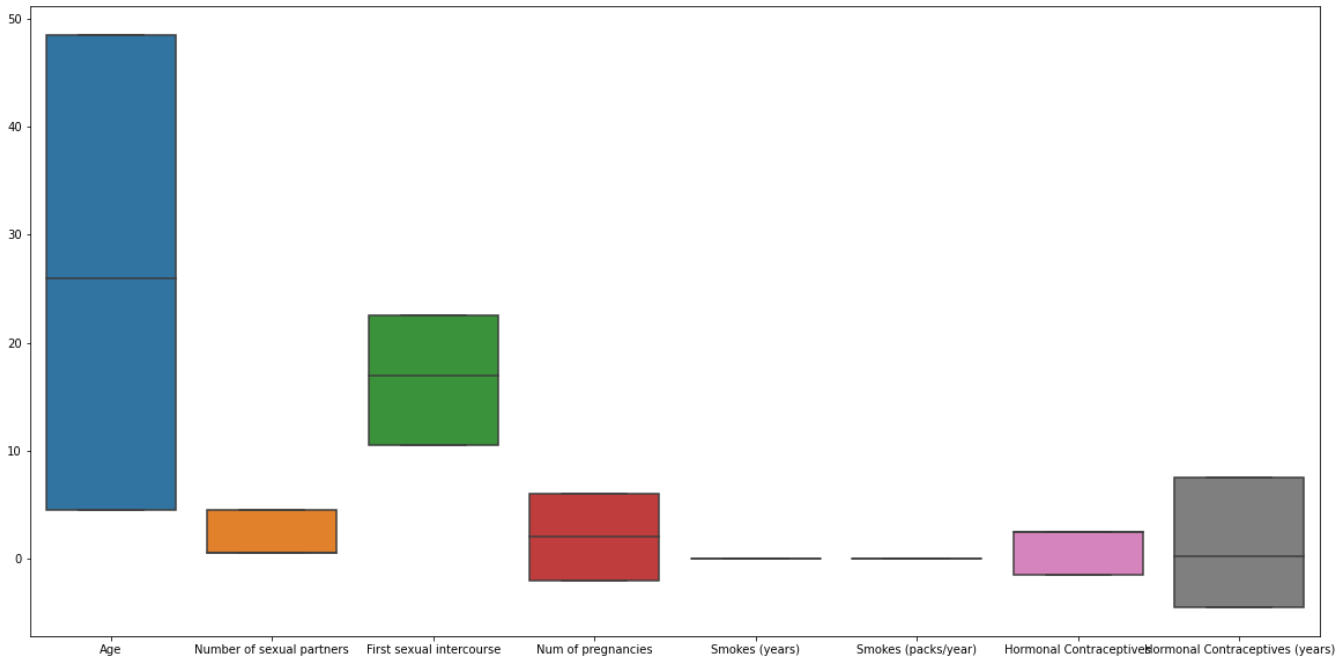
...	...	...	...	...	...	...	...	...	...	...	...
853	48.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
854	48.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
855	25.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...
856	48.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
857	29.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

835 rows × 34 columns

After the treatment of outliers, the boxplot below show that the numerical columns contains no outliers values.

In [20]:
```
plt.figure(figsize = (20,10))
sns.boxplot(data=data[numerical_columns])
```

Out[20]:
```
<AxesSubplot:>
```



In [21]:
```
data.describe()
```

Out[21]:

	Age	STDs: Number of diagnosis	Dx:Cancer	Dx:CIN	Dx:HPV	Dx	Hinselmann	Schiller	
count	835.000000	835.000000	835.000000	835.000000	835.000000	835.000000	835.000000	835.000000	8
mean	25.589222	0.089820	0.021557	0.010778	0.021557	0.028743	0.041916	0.087425	
std	16.664182	0.306335	0.145319	0.103320	0.145319	0.167182	0.200518	0.282626	
min	4.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	4.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	26.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	48.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
max	48.500000	3.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 34 columns

'Biopsy' is the target variable here,the value

0 represent there is no need no biopsy and person doesnt have cervical cancer

1 represent ther is need of biopsy meaning the person has high chances of cervical cancer

In [22]:
```
data['Biopsy'].value_counts()/len(data)
```

Out[22]:
```
0.0    0.935329
1.0    0.064671
Name: Biopsy, dtype: float64
```

6) Dividing the dataset into independent and dependent(target variables)

In [23]:
```
x = data.drop(columns = ['Biopsy'])
y = data['Biopsy']
```

7) Scaling the Dataset

Different columns contains values in different range, if a column contain large values and other contains smaller values like between 0 to 1, the model learns more from the columns which has larger values and has very little or no affect of the columns with smaller values.

Hence, the datset is scaled and the all the values are scaled to a range of (0,1). Doing so all the independent columns have values in same range.

In [24]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_x = scaler.fit_transform(x)
```

8) Removing the multicollinearity

Some features are completely related to other independent features in the dataset. Such columns can be removed from the dataset.

for ex - if we have a radius column and area column, the area column is completely dependent on the radius values and hence it can be removed from the dataset.

In [25]:
```
corr_matrix = x.corr()
corr_matrix.head()
```

Out[25]:

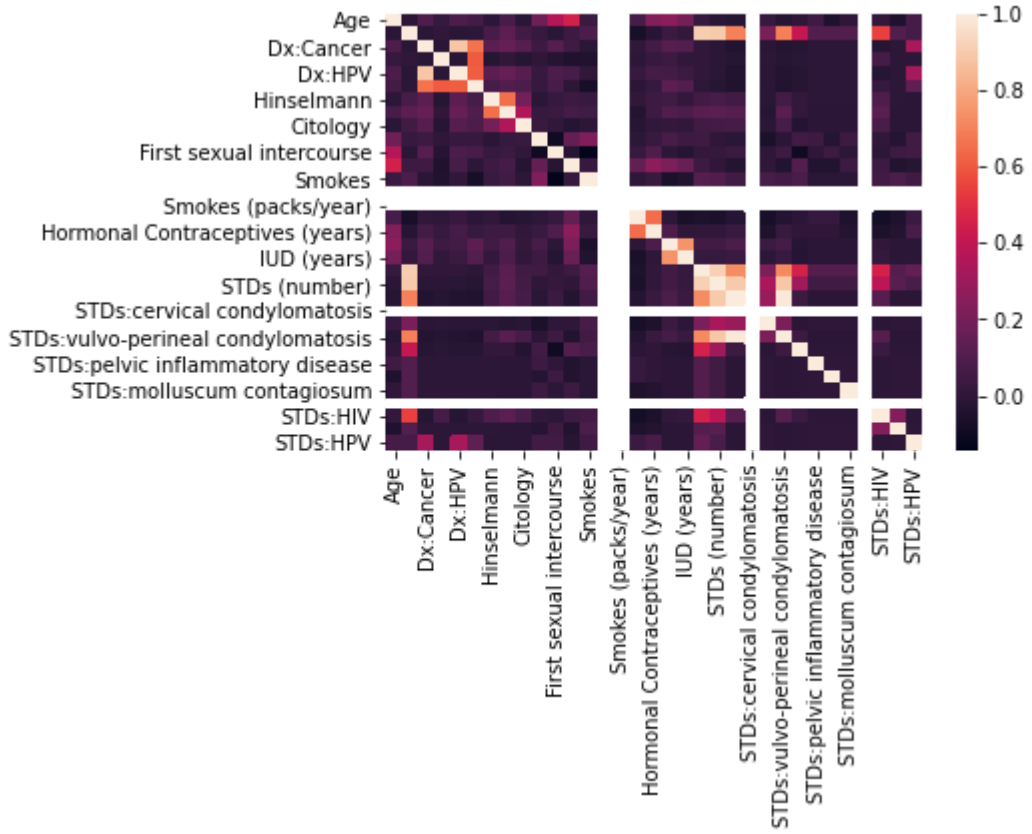
	Age	STDs: Number of diagnosis	Dx:Cancer	Dx:CIN	Dx:HPV	Dx	Hinselmann	Schiller	Citolo
Age	1.000000	-0.011554	0.093034	0.019637	0.088578	0.063421	-0.006324	0.074718	-0.0154
STDs: Number of diagnosis	-0.011554	1.000000	-0.016613	0.007259	-0.016613	-0.003645	0.075276	0.130780	0.0555
Dx:Cancer	0.093034	-0.016613	1.000000	-0.015494	0.886441	0.665423	0.133550	0.158419	0.1146
Dx:CIN	0.019637	0.007259	-0.015494	1.000000	-0.015494	0.606787	-0.021833	0.008753	-0.0243
Dx:HPV	0.088578	-0.016613	0.886441	-0.015494	1.000000	0.616069	0.133550	0.158419	0.1146

5 rows × 33 columns

If the correlation matrix show collinearity more than 0.5 then the column can be removed from the dataset.

Below is the visualisation of that collinearity.

In [26]:
```
sns.heatmap(corr_matrix)
plt.figure(figsize = (10,10),dpi = 150)
plt.show()
```




```
<Figure size 1500x1500 with 0 Axes>
```

Spliting the Dataset

The dataset is split into train set and test set.

The model learn on the train set and the performance of the model is validated using test set.

Train and test set are splitted into 7:3 or 8:2 ratio.

In [27]:
```
from sklearn.model_selection import train_test_split as tts
x_train, x_test, y_train,y_test = tts(scaled_x, y, train_size =
0.7,stratify = None)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[27]:
```
((584, 33), (251, 33), (584,), (251,))
```

Classification Models

1) Logistic Regression

In [28]:
```
from sklearn.linear_model import LogisticRegression as LR
classifier = LR(class_weight = 'balanced')
```

In [29]:
```
classifier.fit(x_train,y_train)
```

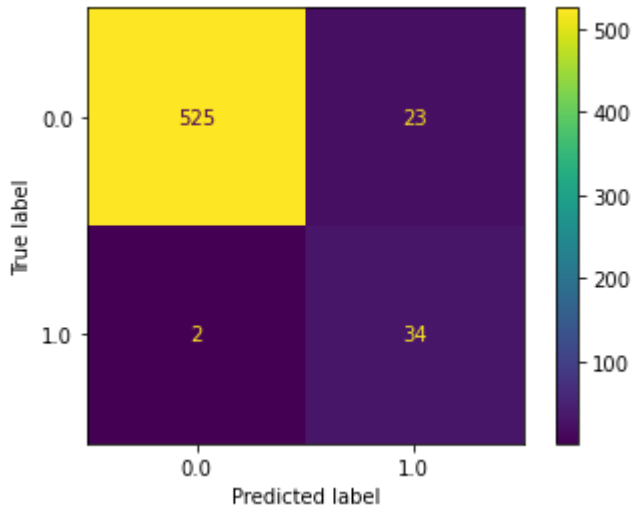
Out[29]:
```
LogisticRegression(class_weight='balanced')
```

y_pred represents the values predicted by the model

In [30]:
```
y_pred = classifier.predict(x_test)
predicted_proba = classifier.predict_proba(x_test)
```

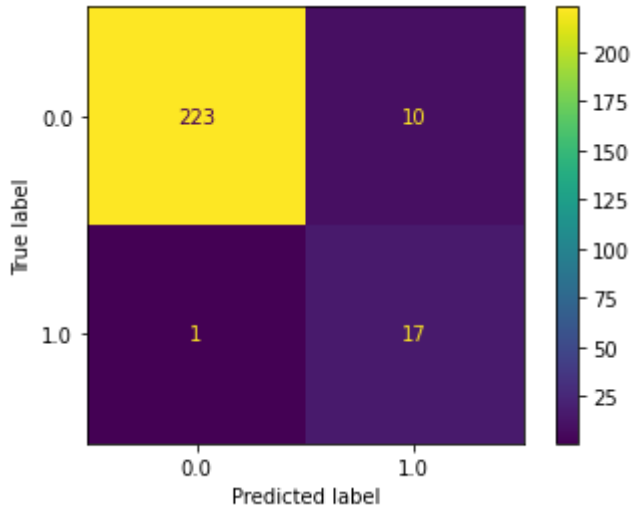
The below confusion matrix show the how well the model can predict the true possitive values while training.

In [31]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, x_train,y_train)
plt.show()
```



The confusion matrix below show how well the model performs on the test set.

In [32]:
```
plot_confusion_matrix(classifier, x_test,y_test)
plt.show()
```



The performance of models build to predict the disease are validated on the basis of its recall score. Recall score lies between 0 to 1. 1 is consider as best model and 0 as the worst model.

The model created by us has a recall value of 0.94 which is a very good score.

In [33]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred)
print(k)
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98       233
```

```
           1.0       0.63      0.94      0.76        18

    accuracy                           0.96       251
   macro avg       0.81      0.95      0.87       251
weighted avg       0.97      0.96      0.96       251
```

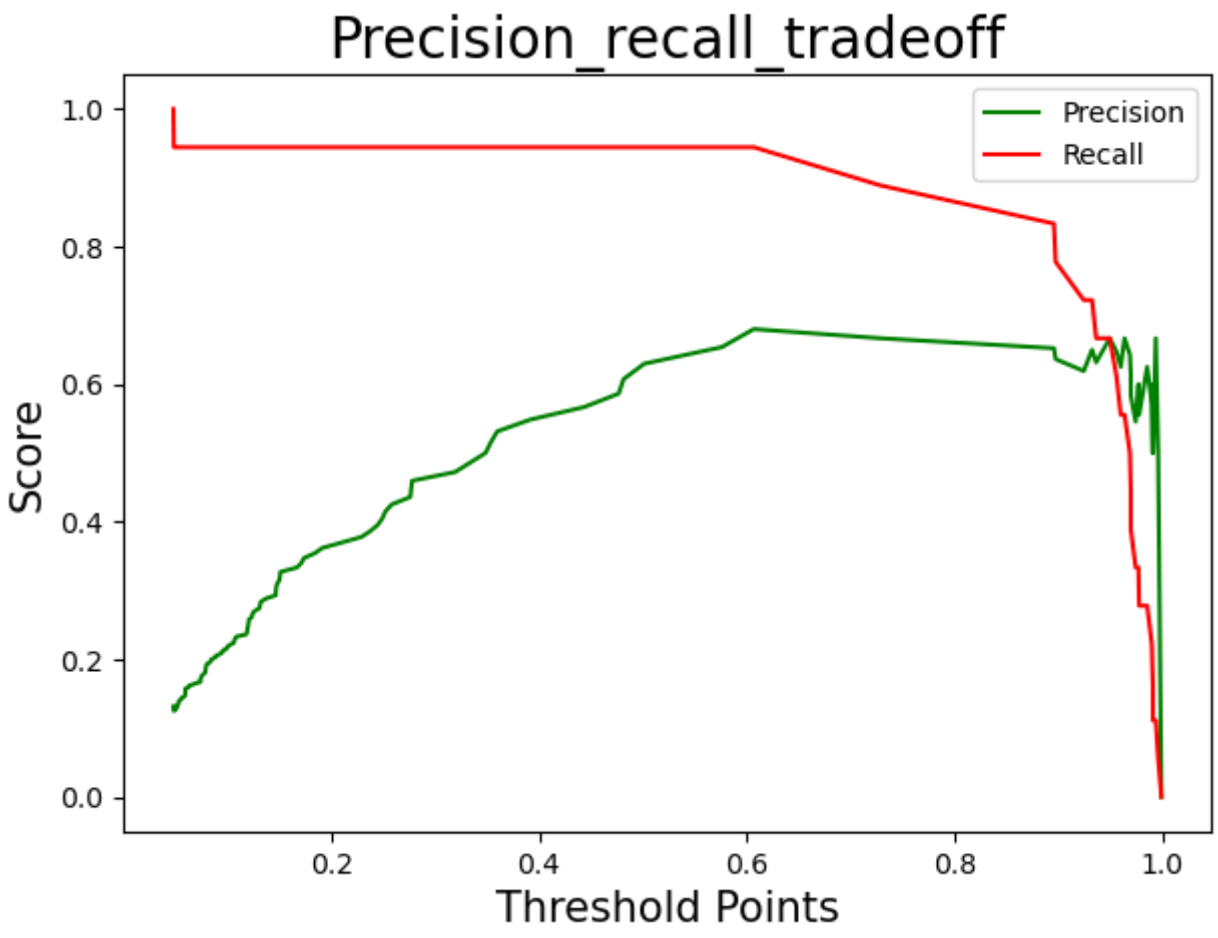
In [34]:
```
from sklearn.metrics import precision_recall_curve
precision_points,recall_points,threshold_points =
precision_recall_curve(y_test,

predicted_proba[:,1])
precision_points.shape,recall_points.shape,threshold_points.shape
```

Out[34]:
```
((127,), (127,), (126,))
```

In [35]:
```
plt.figure(figsize = (7,5),dpi = 100)
plt.plot(threshold_points,precision_points[:-1],color = 'green',label =
'Precision')
plt.plot(threshold_points, recall_points[:-1],color = 'red',label =
'Recall')
plt.xlabel('Threshold Points', fontsize = 15)
plt.ylabel('Score',fontsize = 15)
plt.title('Precision_recall_tradeoff',fontsize = 20)
plt.legend()
```

Out[35]:
```
<matplotlib.legend.Legend at 0x21f22c1e640>
```



In [36]:
```
from sklearn.metrics import roc_curve, roc_auc_score
```

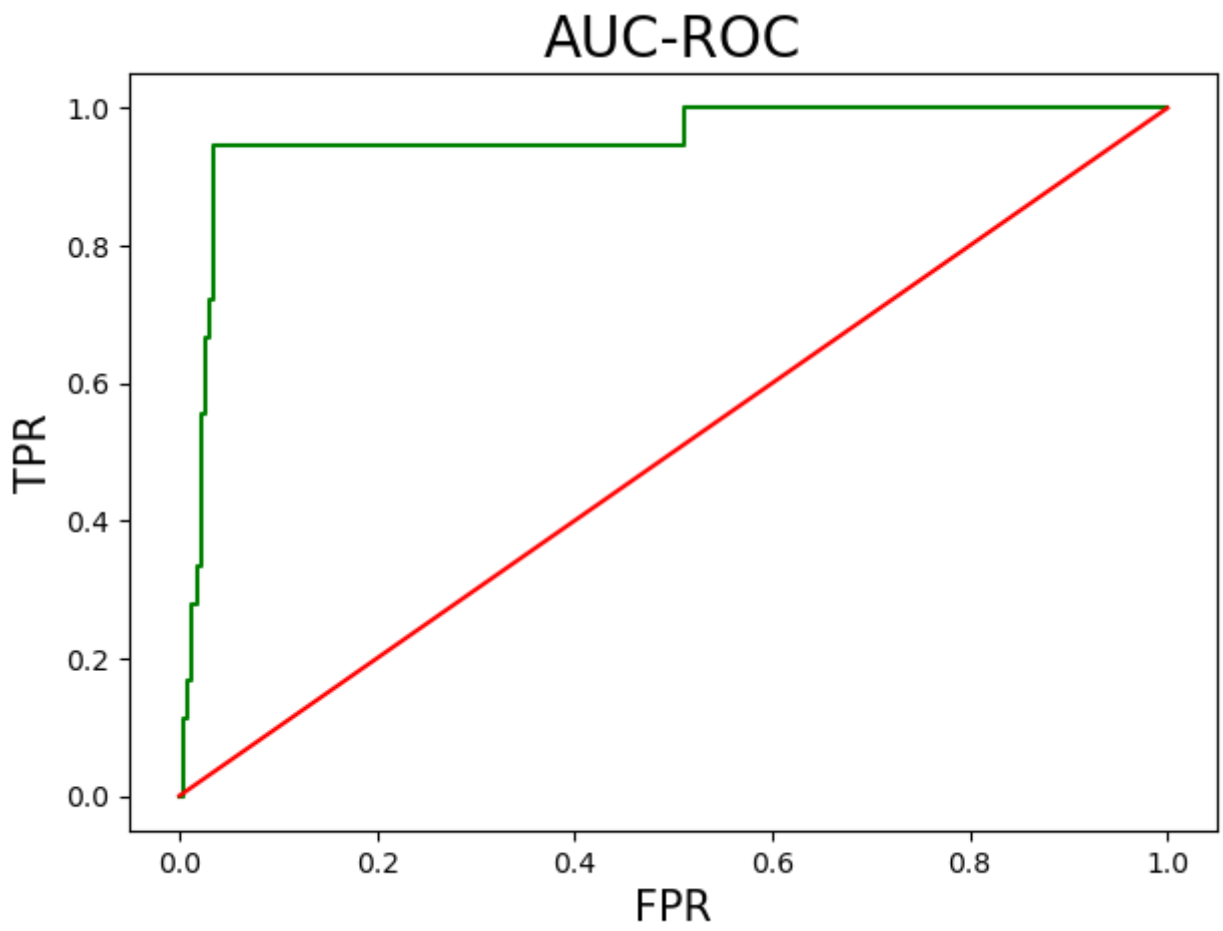
```
fpr,tpr,threshold = roc_curve(y_test,predicted_proba[:,-1])
```

Below is the AOC-Roc curve for the logistic regression model. The values of AUC-ROC lies between 0 to 1. 1 being the best model and 0 being the worst.

The AOC- ROC curve represent how well the model predicts true possitive with respect to its false possitive rate.

The model created by us as AOC- ROC score of 0.96 which is a great score.

In [37]:
```
plt.figure(figsize = (7,5),dpi = 100)
plt.plot(fpr,tpr, color = 'green')
plt.plot([0,1],[0,1],label = 'baseline', color = 'red')
plt.xlabel('FPR', fontsize = 15)
plt.ylabel('TPR', fontsize = 15)
plt.title('AUC-ROC', fontsize = 20)
plt.show()
roc_auc_score(y_test,predicted_proba[:,1])
```



Out[37]:
```
0.9513590844062947
```

The table below show the coefficients generated by the model for each independent variables.

In [38]:
```
c = classifier.coef_.reshape(-1)
x = x.columns
coeff_plot = pd.DataFrame({
    'coefficients':c,
    'variables':x,
    })
```

```
#sorting the values
coeff_plot = coeff_plot.sort_values(by = 'coefficients')
coeff_plot.head()
```

Out[38]:

	coefficients	variables
0	-1.073176	Age
25	-0.634834	STDs:syphilis
10	-0.383990	First sexual intercourse
5	-0.324630	Dx
17	-0.300109	IUD

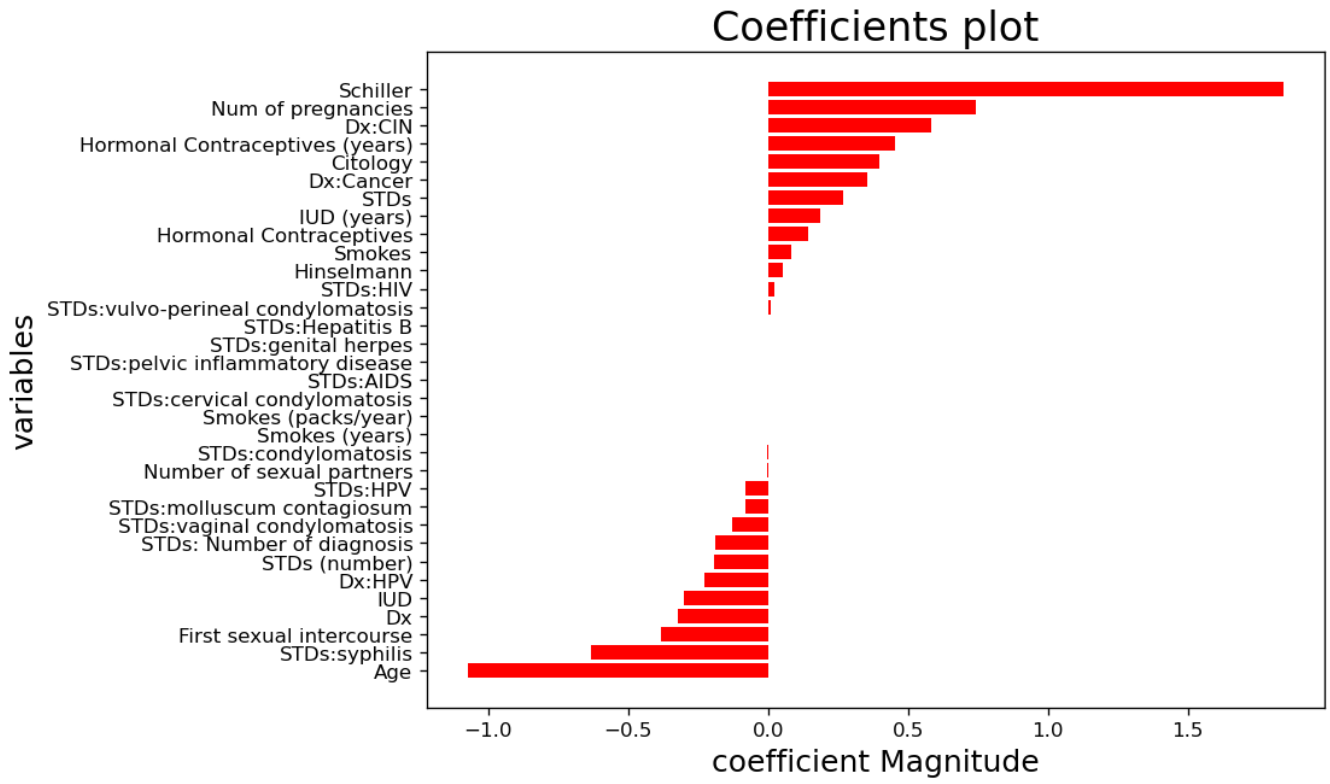
The graph below shows how the target variables depends upon the independent variables used in building this model.

In [39]:

```
plt.figure(figsize = (8,6),dpi = 120)
plt.barh(coeff_plot['variables'],coeff_plot['coefficients'],color =
'red',)
plt.xlabel('coefficient Magnitude',fontsize = 15)
plt.ylabel('variables',fontsize = 15)
plt.title('Coefficients plot',fontsize = 20)
```

Out[39]:

```
Text(0.5, 1.0, 'Coefficients plot')
```



2) Random Forrest

In [40]:

```
from sklearn.ensemble import RandomForestClassifier


rfc = RandomForestClassifier(n_estimators=300)
rfc.fit(x_train, y_train)
y_pred2 = rfc.predict(x_test)
```

In [41]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred2)
print(k)
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97       233
         1.0       0.60      0.67      0.63        18

    accuracy                           0.94       251
   macro avg       0.79      0.82      0.80       251
weighted avg       0.95      0.94      0.95       251
```

3) K- Nearest Neighbours

In [42]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(x_train, y_train)

y_pred3 = knn.predict(x_test)
```

In [43]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred3)
print(k)
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.98      0.96       233
         1.0       0.56      0.28      0.37        18

    accuracy                           0.93       251
   macro avg       0.75      0.63      0.67       251
weighted avg       0.92      0.93      0.92       251
```

4) Support Vector Machine

In [44]:
```
from sklearn.svm import SVC

svc_model = SVC(kernel="rbf")
svc_model.fit(x_train, y_train)
y_pred4 = svc_model.predict(x_test)
```

In [45]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred4)
print(k)
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.98      0.96       233
         1.0       0.50      0.28      0.36        18

    accuracy                           0.93       251
   macro avg       0.72      0.63      0.66       251
weighted avg       0.91      0.93      0.92       251
```

On comparing all these models it is found that the logistic regression model predicts most precisely with a

recall score of 0.94

CHAPTER 6:
MACHINE LEARNING MODELS

BREAST CANCER PREDICTION

Breast Cancer Prediction

1) Importing the necessary Python libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
pd.options.mode.chained_assignment = None
```

2) Importing the dataset

In [2]:
```
data = pd.read_csv('breast-cancer.csv')
data.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 33 columns

Exploratory Data Analysis

3) Data Cleaning

In [3]:
```
data.drop(['id'],axis = 'columns',inplace = True)
```

In [4]:
```
data.head()
```

Out[4]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_me
0	M	17.99	10.38	122.80	1001.0	0.11840	0.277
1	M	20.57	17.77	132.90	1326.0	0.08474	0.078
2	M	19.69	21.25	130.00	1203.0	0.10960	0.159
3	M	11.42	20.38	77.58	386.1	0.14250	0.283
4	M	20.29	14.34	135.10	1297.0	0.10030	0.132

5 rows × 32 columns

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                  Non-Null Count  Dtype
```

```
---  ------                   --------------  -----
 0   diagnosis                569 non-null    object
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
 31  Unnamed: 32              0 non-null      float64
dtypes: float64(31), object(1)
memory usage: 142.4+ KB
```

In [6]:
```
data.drop(['Unnamed: 32'],axis = 'columns',inplace = True)
```

In [7]:
```
data['diagnosis'].value_counts()/len(data)
```

Out[7]:
```
B    0.627417
M    0.372583
Name: diagnosis, dtype: float64
```

In [8]:
```
data['diagnosis'].unique()
```

Out[8]:
```
array(['M', 'B'], dtype=object)
```

4) Variable Transformation

In [9]:
```
mapping = {'M':'1',
           'B':'0'}
data['diagnosis'] = data['diagnosis'].map(mapping)
```

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

Out[10]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_me
0	1	17.99	10.38	122.80	1001.0	0.11840	0.277
1	1	20.57	17.77	132.90	1326.0	0.08474	0.078
2	1	19.69	21.25	130.00	1203.0	0.10960	0.159
3	1	11.42	20.38	77.58	386.1	0.14250	0.283

4	1	20.29	14.34	135.10	1297.0	0.10030	0.132

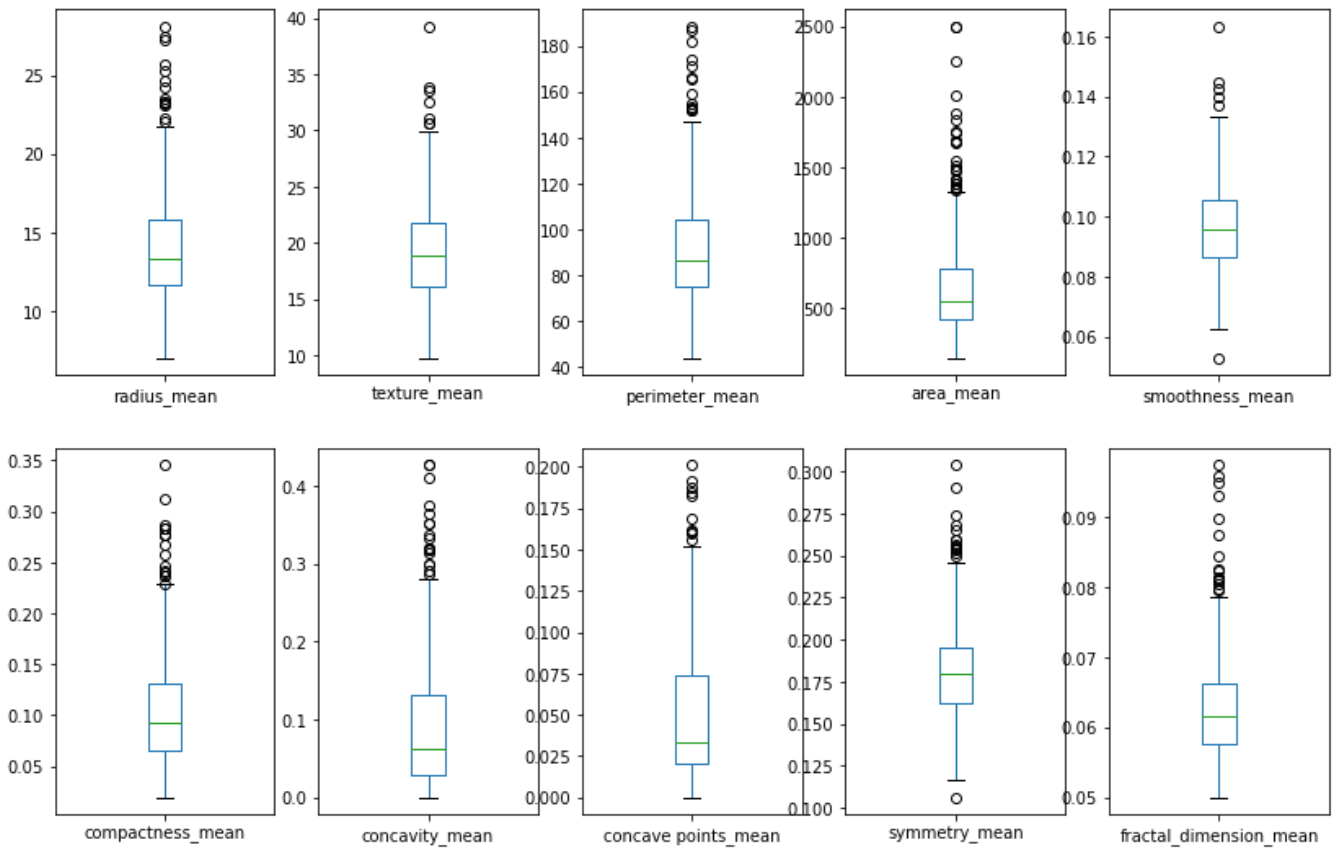
5 rows × 31 columns

In [11]:
```
numerical= ['radius_mean', 'texture_mean', 'perimeter_mean',
'area_mean',
        'smoothness_mean', 'compactness_mean', 'concavity_mean',
        'concave points_mean', 'symmetry_mean',
'fractal_dimension_mean',
        'radius_se', 'texture_se', 'perimeter_se', 'area_se',
'smoothness_se',
        'compactness_se', 'concavity_se', 'concave points_se',
'symmetry_se',
        'fractal_dimension_se', 'radius_worst', 'texture_worst',
        'perimeter_worst', 'area_worst', 'smoothness_worst',
        'compactness_worst', 'concavity_worst', 'concave points_worst',
        'symmetry_worst', 'fractal_dimension_worst']
```

5) Treating the outliers

In [12]:
```
data[numerical[0:10]].plot(kind = 'box', subplots = True,
                           layout = (3,5),fontsize = 10,
                           figsize = (14,14))
```

Out[12]:
```
radius_mean                 AxesSubplot(0.125,0.657941;0.133621x0.222059)
texture_mean              AxesSubplot(0.285345,0.657941;0.133621x0.222059)
perimeter_mean             AxesSubplot(0.44569,0.657941;0.133621x0.222059)
area_mean                 AxesSubplot(0.606034,0.657941;0.133621x0.222059)
smoothness_mean           AxesSubplot(0.766379,0.657941;0.133621x0.222059)
compactness_mean             AxesSubplot(0.125,0.391471;0.133621x0.222059)
concavity_mean            AxesSubplot(0.285345,0.391471;0.133621x0.222059)
concave points_mean        AxesSubplot(0.44569,0.391471;0.133621x0.222059)
symmetry_mean             AxesSubplot(0.606034,0.391471;0.133621x0.222059)
fractal_dimension_mean    AxesSubplot(0.766379,0.391471;0.133621x0.222059)
dtype: object
```



In [13]:
```
data.describe()
```

Out[13]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	c
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	

8 rows × 30 columns

In [14]:
```
x = data.drop(columns = ['diagnosis'])
y = data['diagnosis']
```

In [15]:
```
def cap_data(data):
    for col in numerical:
        if (((data[col].dtype)=='float64') |
((data[col].dtype)=='int64')):
            percentiles = data[col].quantile([0.25,0.75]).values
            iqr = percentiles[1]-percentiles[0]
            upper_limit = percentiles[1] + 1.5*iqr
            lower_limit = percentiles[0] -1.5*iqr
            data[col][data[col] <= percentiles[0]] = lower_limit
            data[col][data[col] >= percentiles[1]] = upper_limit
```

```
        else:
            data[col]=data[col]
    return data


cap_data(data)
```

Out[15]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_
0	1	21.90	7.725	147.495	1326.3	0.133695	0.2
1	1	21.90	17.770	147.495	1326.3	0.057975	0.0
2	1	21.90	21.250	147.495	1326.3	0.133695	0.2
3	1	5.58	20.380	77.580	-123.3	0.133695	0.2
4	1	21.90	7.725	147.495	1326.3	0.100300	0.2
...	...	...	...	...	...	...	
564	1	21.90	30.245	147.495	1326.3	0.133695	0.1
565	1	21.90	30.245	147.495	1326.3	0.097800	0.1
566	1	21.90	30.245	147.495	1326.3	0.057975	0.1
567	1	21.90	30.245	147.495	1326.3	0.133695	0.2
568	0	5.58	30.245	31.775	-123.3	0.057975	-0.0

569 rows × 31 columns

In [16]:

```
data.head()
```

Out[16]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_me
0	1	21.90	7.725	147.495	1326.3	0.133695	0.228
1	1	21.90	17.770	147.495	1326.3	0.057975	0.078
2	1	21.90	21.250	147.495	1326.3	0.133695	0.228
3	1	5.58	20.380	77.580	-123.3	0.133695	0.228
4	1	21.90	7.725	147.495	1326.3	0.100300	0.228

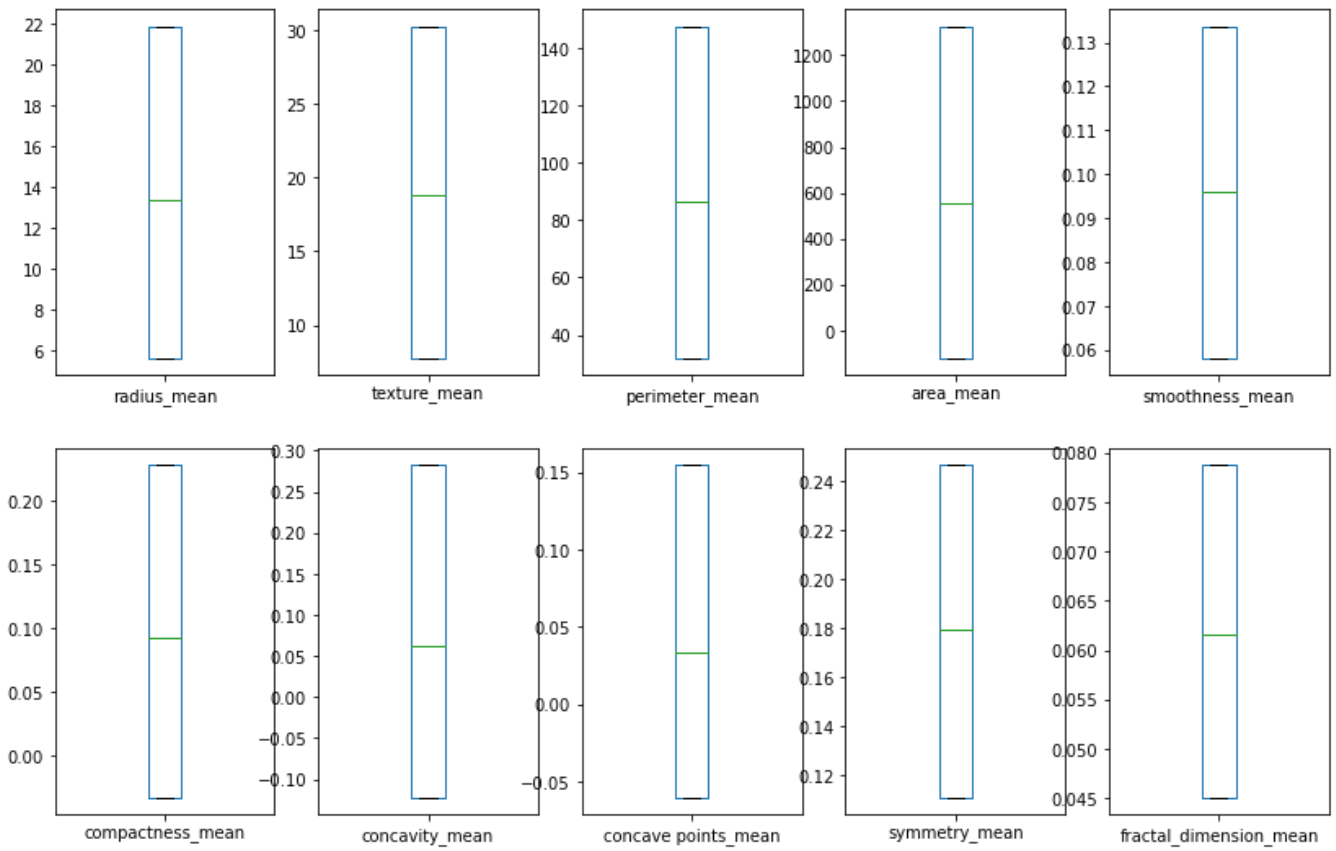
5 rows × 31 columns

In [17]:

```
data[numerical[0:10]].plot(kind = 'box', subplots = True,
                           layout = (3,5),fontsize = 10,
                           figsize = (14,14))
```

Out[17]:

```
radius_mean                AxesSubplot(0.125,0.657941;0.133621x0.222059)
texture_mean             AxesSubplot(0.285345,0.657941;0.133621x0.222059)
perimeter_mean            AxesSubplot(0.44569,0.657941;0.133621x0.222059)
area_mean                AxesSubplot(0.606034,0.657941;0.133621x0.222059)
smoothness_mean          AxesSubplot(0.766379,0.657941;0.133621x0.222059)
compactness_mean           AxesSubplot(0.125,0.391471;0.133621x0.222059)
concavity_mean           AxesSubplot(0.285345,0.391471;0.133621x0.222059)
concave points_mean       AxesSubplot(0.44569,0.391471;0.133621x0.222059)
symmetry_mean            AxesSubplot(0.606034,0.391471;0.133621x0.222059)
fractal_dimension_mean   AxesSubplot(0.766379,0.391471;0.133621x0.222059)
dtype: object
```



In [18]:
```
data.head(10)
```

Out[18]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_me
0	1	21.90	7.725	147.495	1326.3	0.133695	0.228
1	1	21.90	17.770	147.495	1326.3	0.057975	0.078
2	1	21.90	21.250	147.495	1326.3	0.133695	0.228
3	1	5.58	20.380	77.580	-123.3	0.133695	0.228
4	1	21.90	7.725	147.495	1326.3	0.100300	0.228
5	1	12.45	7.725	82.570	477.1	0.133695	0.228
6	1	21.90	19.980	147.495	1326.3	0.094630	0.109
7	1	13.71	20.830	90.200	577.9	0.133695	0.228
8	1	13.00	30.245	87.500	519.8	0.133695	0.228
9	1	12.46	30.245	83.970	475.9	0.133695	0.228

10 rows × 31 columns

In [19]:
```
data.describe()
```

Out[19]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	c
count	569.00000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	13.61065	18.961248	88.423989	580.945694	0.095814	0.096439	
std	5.85459	8.066538	41.445420	519.796264	0.027181	0.094079	
min	5.58000	7.725000	31.775000	-123.300000	0.057975	-0.033300	
25%	5.58000	7.725000	31.775000	-123.300000	0.057975	-0.033300	
50%	13.37000	18.840000	86.240000	551.100000	0.095870	0.092630	
75%	21.90000	30.245000	147.495000	1326.300000	0.133695	0.228620	

max	21.90000	30.245000	147.495000	1326.300000	0.133695	0.228620

8 rows × 30 columns

In [20]:
```
x = data.drop(columns = ['diagnosis'])
y = data['diagnosis']
```

6) Scaling the data set

In [21]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_x = scaler.fit_transform(x)
```

7) Treating multicollinearity

In [22]:
```
corr_matrix = x.corr()
corr_matrix.head()
```

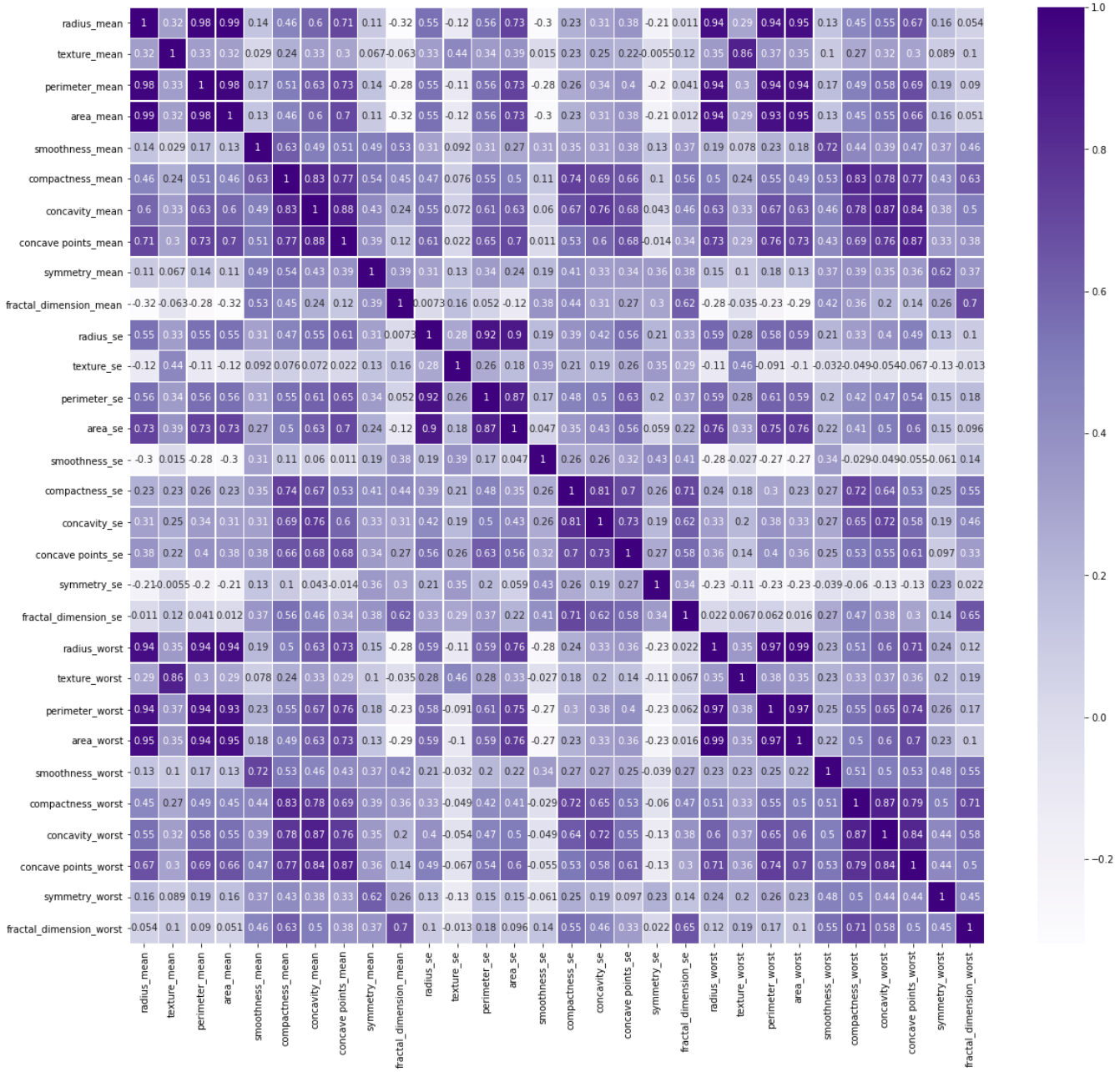
Out[22]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactnes
radius_mean	1.000000	0.323173	0.982896	0.992067	0.137031	
texture_mean	0.323173	1.000000	0.328690	0.323070	0.029284	
perimeter_mean	0.982896	0.328690	1.000000	0.982629	0.172396	
area_mean	0.992067	0.323070	0.982629	1.000000	0.131460	
smoothness_mean	0.137031	0.029284	0.172396	0.131460	1.000000	

5 rows × 30 columns

In [23]:
```
plt.figure(figsize=(20,18))
sns.heatmap(data.corr(), annot = True, linewidths = .5, cmap =
'Purples')
```

Out[23]:
```
<AxesSubplot:>
```



In [24]:

```
#vif
from statsmodels.stats.outliers_influence import
variance_inflation_factor
vif_data = x
VIF = pd.Series([variance_inflation_factor(vif_data.values,i) for i in
range(vif_data.shape[1])],
                index = vif_data.columns)
VIF
```

Out[24]:

```
radius_mean                331.715621
texture_mean                30.711086
perimeter_mean             214.510607
area_mean                  117.092525
smoothness_mean             50.272888
compactness_mean            15.657516
concavity_mean              13.331857
concave points_mean         11.470793
symmetry_mean               37.439662
fractal_dimension_mean      97.006020
radius_se                   20.471780
texture_se                   6.644452
perimeter_se                16.780030
area_se                     15.805668
smoothness_se                8.950256
compactness_se               9.671503
```

```
concavity_se                8.276281
concave points_se           9.543037
symmetry_se                 9.133005
fractal_dimension_se        8.148298
radius_worst              224.500091
texture_worst              33.402518
perimeter_worst           105.985763
area_worst                 95.905683
smoothness_worst           47.698063
compactness_worst          15.424702
concavity_worst            13.193150
concave points_worst       13.510635
symmetry_worst             33.812673
fractal_dimension_worst    48.418545
dtype: float64
```

In [25]:
```
VIF[VIF == VIF.max()].index[0]
```

Out[25]:
```
'radius_mean'
```

Spliting the Dataset

In [26]:
```
from sklearn.model_selection import train_test_split as tts
x_train, x_test, y_train,y_test = tts(scaled_x, y, train_size =
0.7,stratify = None)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[26]:
```
((398, 30), (171, 30), (398,), (171,))
```

Classification Models

1) Logistic Regresssion

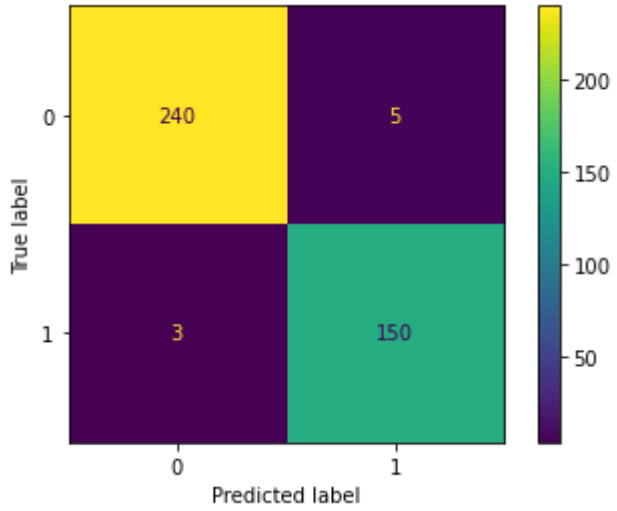
In [27]:
```
from sklearn.linear_model import LogisticRegression as LR
classifier = LR(class_weight = 'balanced')
```

In [28]:
```
classifier.fit(x_train,y_train)
```

Out[28]:
```
LogisticRegression(class_weight='balanced')
```

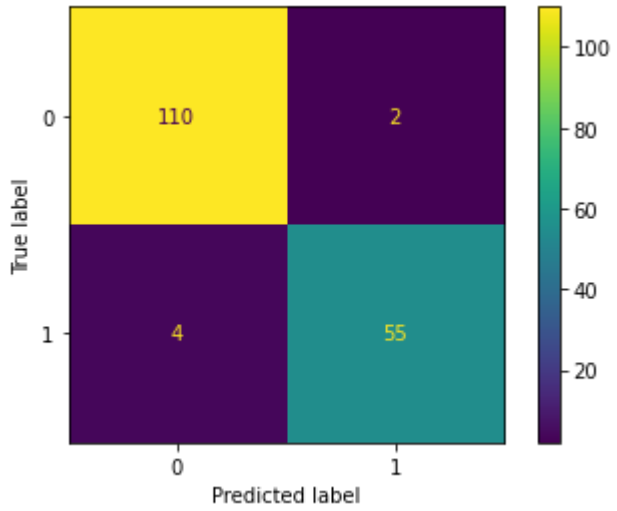
In [29]:
```
y_pred1 = classifier.predict(x_test)
predicted_proba = classifier.predict_proba(x_test)
```

In [30]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, x_train,y_train)
plt.show()
```



In [31]:

```
plot_confusion_matrix(classifier, x_test,y_test)
plt.show()
```



In [32]:

```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred1)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       112
           1       0.96      0.93      0.95        59

    accuracy                           0.96       171
   macro avg       0.96      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

In [33]:

```
from sklearn.metrics import precision_recall_curve
precision_points,recall_points,threshold_points =
precision_recall_curve(y_test,

predicted_proba[:,1],pos_label ='1')
precision_points.shape,recall_points.shape,threshold_points.shape
```

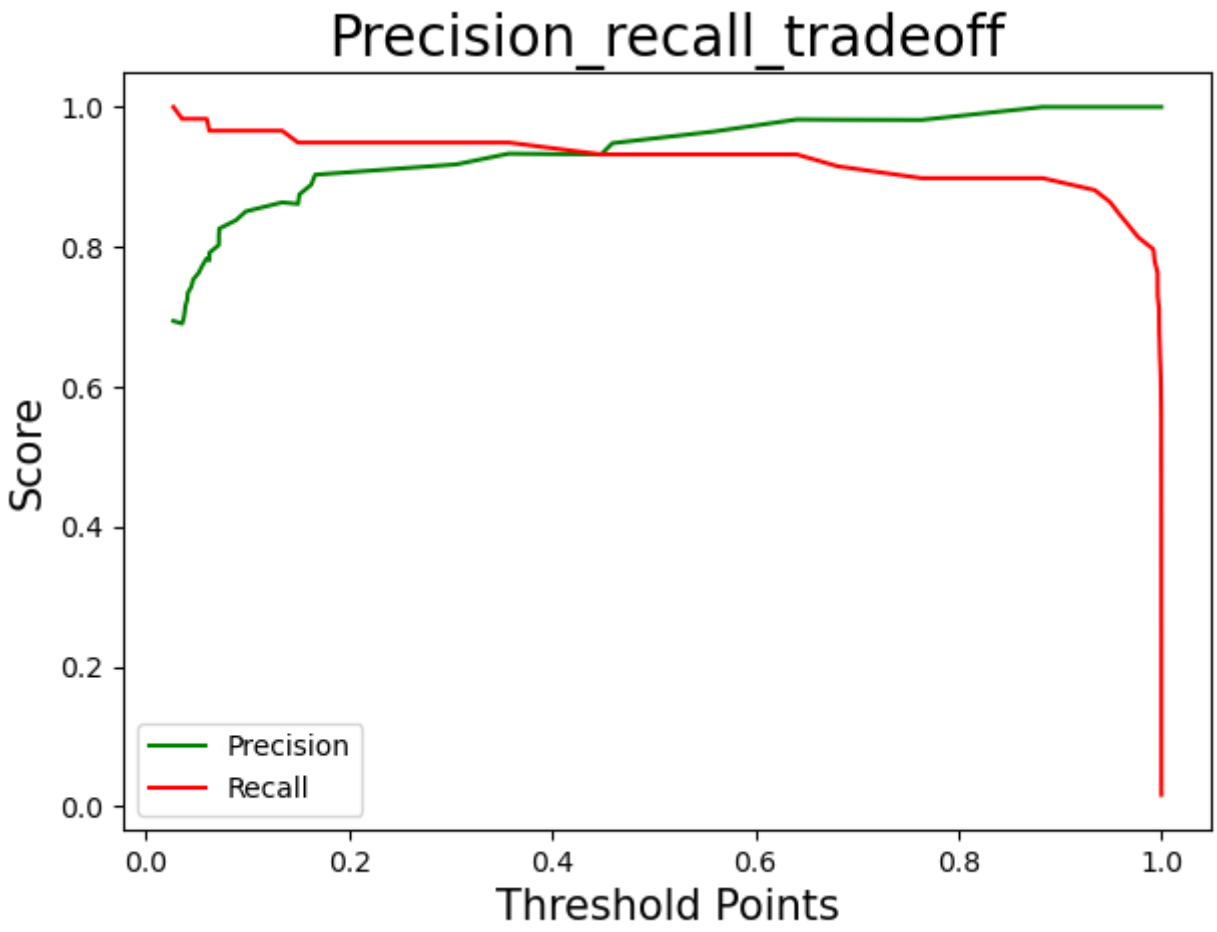
Out[33]:

```
((86,), (86,), (85,))
```

Precision- Recall Tradeoff

In [34]:
```
plt.figure(figsize = (7,5),dpi = 100)
plt.plot(threshold_points,precision_points[:-1],color = 'green',label =
'Precision')
plt.plot(threshold_points, recall_points[:-1],color = 'red',label =
'Recall')
plt.xlabel('Threshold Points', fontsize = 15)
plt.ylabel('Score',fontsize = 15)
plt.title('Precision_recall_tradeoff',fontsize = 20)
plt.legend()
```

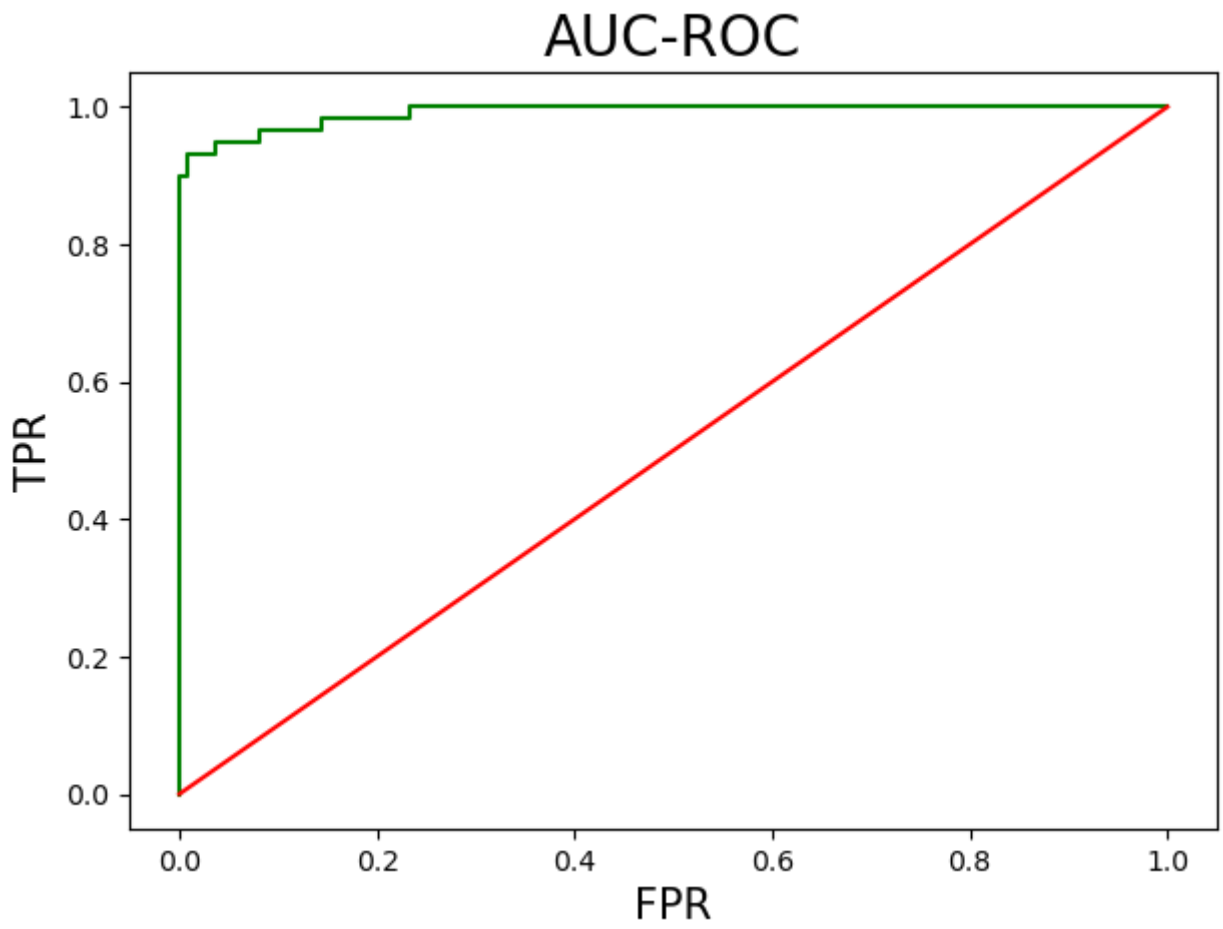
Out[34]:
```
<matplotlib.legend.Legend at 0x15363742670>
```



In [35]:
```
from sklearn.metrics import roc_curve, roc_auc_score
fpr,tpr,threshold = roc_curve(y_test,predicted_proba[:,-1],pos_label =
'1')
```

Aoc - RoC CUrve

In [36]:
```
plt.figure(figsize = (7,5),dpi = 100)
plt.plot(fpr,tpr, color = 'green')
plt.plot([0,1],[0,1],label = 'baseline', color = 'red')
plt.xlabel('FPR', fontsize = 15)
plt.ylabel('TPR', fontsize = 15)
plt.title('AUC-ROC', fontsize = 20)
plt.show()
roc_auc_score(y_test,predicted_proba[:,1])
```



Out[36]:
```
0.9913740920096852
```

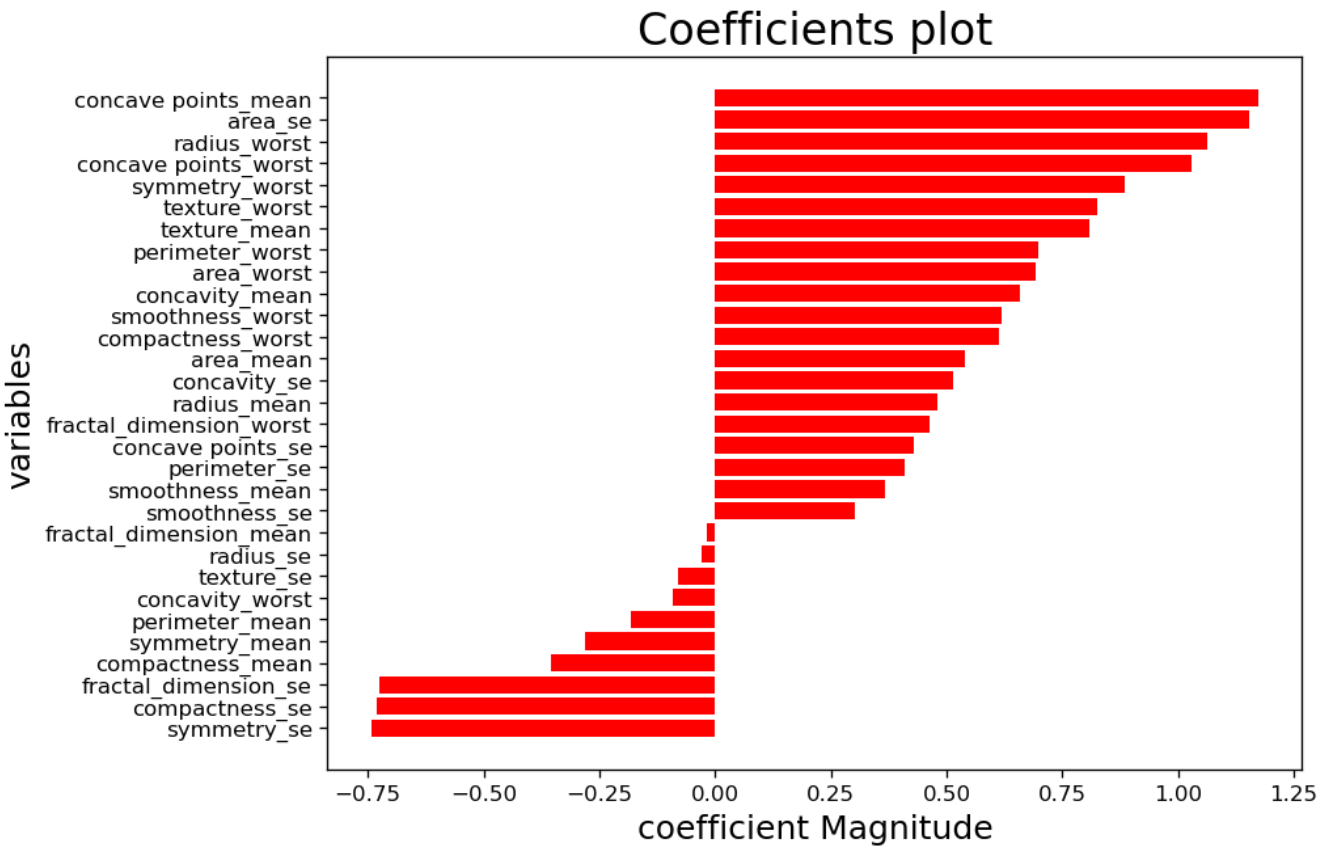
In [37]:
```
c = classifier.coef_.reshape(-1)
x = x.columns
coeff_plot = pd.DataFrame({
    'coefficients':c,
    'variables':x,
    })
#sorting the values
coeff_plot = coeff_plot.sort_values(by = 'coefficients')
coeff_plot.head()
```

Out[37]:

	coefficients	variables
18	-0.741155	symmetry_se
15	-0.731980	compactness_se
19	-0.724192	fractal_dimension_se
5	-0.353039	compactness_mean
8	-0.280754	symmetry_mean

In [38]:
```
plt.figure(figsize = (8,6),dpi = 120)
plt.barh(coeff_plot['variables'],coeff_plot['coefficients'],color =
'red')
plt.xlabel('coefficient Magnitude',fontsize = 15)
plt.ylabel('variables',fontsize = 15)
plt.title('Coefficients plot',fontsize = 20)
```

Out[38]:
```
Text(0.5, 1.0, 'Coefficients plot')
```



2)Random Forest

In [39]:
```
from sklearn.ensemble import RandomForestClassifier


rfc = RandomForestClassifier(n_estimators=300)
rfc.fit(x_train, y_train)
y_pred2 = rfc.predict(x_test)
```

In [40]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred2)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.96       112
           1       0.95      0.88      0.91        59

    accuracy                           0.94       171
   macro avg       0.94      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171
```

3)K-Nearest Neighbors

In [41]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)


knn.fit(x_train, y_train)


y_pred3 = knn.predict(x_test)
```

In [42]:
```
from sklearn.metrics import classification_report
```

```
k = classification_report(y_test,y_pred3)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.93      0.99      0.96       112
           1       0.98      0.86      0.92        59

    accuracy                           0.95       171
   macro avg       0.96      0.93      0.94       171
weighted avg       0.95      0.95      0.95       171
```

4)Support Vector Machines

In [43]:
```
from sklearn.svm import SVC


svc_model = SVC(kernel="rbf")
svc_model.fit(x_train, y_train)
y_pred4 = svc_model.predict(x_test)
```

In [44]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred4)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       112
           1       0.96      0.90      0.93        59

    accuracy                           0.95       171
   macro avg       0.96      0.94      0.95       171
weighted avg       0.95      0.95      0.95       171
```

On comparing all these models it is found that the support vector machine model predicts most precisely with a recall score of (0.98,0.93)

CHAPTER 6:
MACHINE LEARNING MODELS

ANEMIA PREDICTION

Anemia Prediction

1) Importing the necessary Python libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
pd.options.mode.chained_assignment = None
```

2) Importing the dataset

In [2]:
```
data = pd.read_csv('Anemia.csv')
data.head(10)
```

Out[2]:

	NO	LAB_TEST	RESULT	REFERENCE_INTERVAL	GENDER	IDENTIFICATION
0	1	Hemoglobin	14.9	13.5 - 17.5	Male	Not Anemia
1	1	MCH	22.7	27.0 - 31.0	Male	Not Anemia
2	1	MCHC	29.1	32.0 - 36.0	Male	Not Anemia
3	1	MCV	83.7	82.0 - 92.0	Male	Not Anemia
4	2	Hemoglobin	15.9	12.0 - 16.0	Female	Not Anemia
5	2	MCH	25.4	27.0 - 31.0	Female	Not Anemia
6	2	MCHC	28.3	32.0 - 36.0	Female	Not Anemia
7	2	MCV	72.0	82.0 - 92.0	Female	Not Anemia
8	3	Hemoglobin	9.0	12.0 - 16.0	Female	Anemia
9	3	MCH	21.5	27.0 - 31.0	Female	Anemia

Exploratory Data Analysis

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5684 entries, 0 to 5683
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   NO                  5684 non-null   int64  
 1   LAB_TEST            5684 non-null   object 
 2   RESULT              5684 non-null   float64
 3   REFERENCE_INTERVAL  5684 non-null   object 
 4   GENDER              5684 non-null   object 
 5   IDENTIFICATION      5684 non-null   object 
dtypes: float64(1), int64(1), object(4)
memory usage: 266.6+ KB
```

In [4]:
```
data.shape
```

Out[4]:
```
(5684, 6)
```

In [5]:
```
data.columns
```

Out[5]:
```
Index(['NO', 'LAB_TEST', 'RESULT', 'REFERENCE_INTERVAL', 'GENDER',
       'IDENTIFICATION'],
      dtype='object')
```

3) Data Cleaning

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
NO                    0
LAB_TEST              0
RESULT                0
REFERENCE_INTERVAL    0
GENDER                0
IDENTIFICATION        0
dtype: int64
```

In [7]:
```
data['LAB_TEST'].unique()
```

Out[7]:
```
array(['Hemoglobin', 'MCH', 'MCHC', 'MCV'], dtype=object)
```

In [8]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
data['LAB_TEST'] = label_encoder.fit_transform(data['LAB_TEST'])
data['LAB_TEST'].unique()
```

Out[8]:
```
array([0, 1, 2, 3])
```

In [9]:
```
label_encoder = LabelEncoder()
data['GENDER'] = label_encoder.fit_transform(data['GENDER'])
data['GENDER'].unique()
```

Out[9]:
```
array([1, 0])
```

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

Out[10]:

	NO	LAB_TEST	RESULT	REFERENCE_INTERVAL	GENDER	IDENTIFICATION
0	1	0	14.9	13.5 - 17.5	1	Not Anemia
1	1	1	22.7	27.0 - 31.0	1	Not Anemia
2	1	2	29.1	32.0 - 36.0	1	Not Anemia
3	1	3	83.7	82.0 - 92.0	1	Not Anemia
4	2	0	15.9	12.0 - 16.0	0	Not Anemia

In [11]:
```
data.drop('NO',axis = 'columns',inplace = True)
```

In [12]:
```
data.head()
```

Out[12]:

	LAB_TEST	RESULT	REFERENCE_INTERVAL	GENDER	IDENTIFICATION
0	0	14.9	13.5 - 17.5	1	Not Anemia
1	1	22.7	27.0 - 31.0	1	Not Anemia
2	2	29.1	32.0 - 36.0	1	Not Anemia
3	3	83.7	82.0 - 92.0	1	Not Anemia
4	0	15.9	12.0 - 16.0	0	Not Anemia

In [13]:
```
mapping = {'Not Anemia':'0',
           'Anemia':'1'}
data['IDENTIFICATION'] = data['IDENTIFICATION'].map(mapping)
```

In [14]:
```
data['IDENTIFICATION'].value_counts()/len(data)
```

Out[14]:
```
0    0.563688
1    0.436312
Name: IDENTIFICATION, dtype: float64
```

In [15]:
```
data.head()
```

Out[15]:

	LAB_TEST	RESULT	REFERENCE_INTERVAL	GENDER	IDENTIFICATION
0	0	14.9	13.5 - 17.5	1	0
1	1	22.7	27.0 - 31.0	1	0
2	2	29.1	32.0 - 36.0	1	0
3	3	83.7	82.0 - 92.0	1	0
4	0	15.9	12.0 - 16.0	0	0

In [16]:
```
df = data['REFERENCE_INTERVAL']
data[['Lower_reference_limit','Upper_reference_limit']] =
df.str.split("-",expand = True)
```

In [17]:
```
data.head()
```

Out[17]:

	LAB_TEST	RESULT	REFERENCE_INTERVAL	GENDER	IDENTIFICATION	Lower_reference_limit	Upper_r
0	0	14.9	13.5 - 17.5	1	0	13.5	
1	1	22.7	27.0 - 31.0	1	0	27.0	
2	2	29.1	32.0 - 36.0	1	0	32.0	
3	3	83.7	82.0 - 92.0	1	0	82.0	
4	0	15.9	12.0 - 16.0	0	0	12.0	

In [18]:
```
data.drop('REFERENCE_INTERVAL',axis = 'columns', inplace = True)
```

In [19]:
```
data.head()
```

Out[19]:

	LAB_TEST	RESULT	GENDER	IDENTIFICATION	Lower_reference_limit	Upper_reference_limit
0	0	14.9	1	0	13.5	17.5
1	1	22.7	1	0	27.0	31.0
2	2	29.1	1	0	32.0	36.0
3	3	83.7	1	0	82.0	92.0
4	0	15.9	0	0	12.0	16.0

In [20]:
```
column_names =
['LAB_TEST','GENDER','Lower_reference_limit','Upper_reference_limit','RES
```

```
                    'IDENTIFICATION']
data = data.reindex(columns = column_names)
```

In [21]:
```
data.head()
```

Out[21]:

	LAB_TEST	GENDER	Lower_reference_limit	Upper_reference_limit	RESULT	IDENTIFICATION
0	0	1	13.5	17.5	14.9	0
1	1	1	27.0	31.0	22.7	0
2	2	1	32.0	36.0	29.1	0
3	3	1	82.0	92.0	83.7	0
4	0	0	12.0	16.0	15.9	0

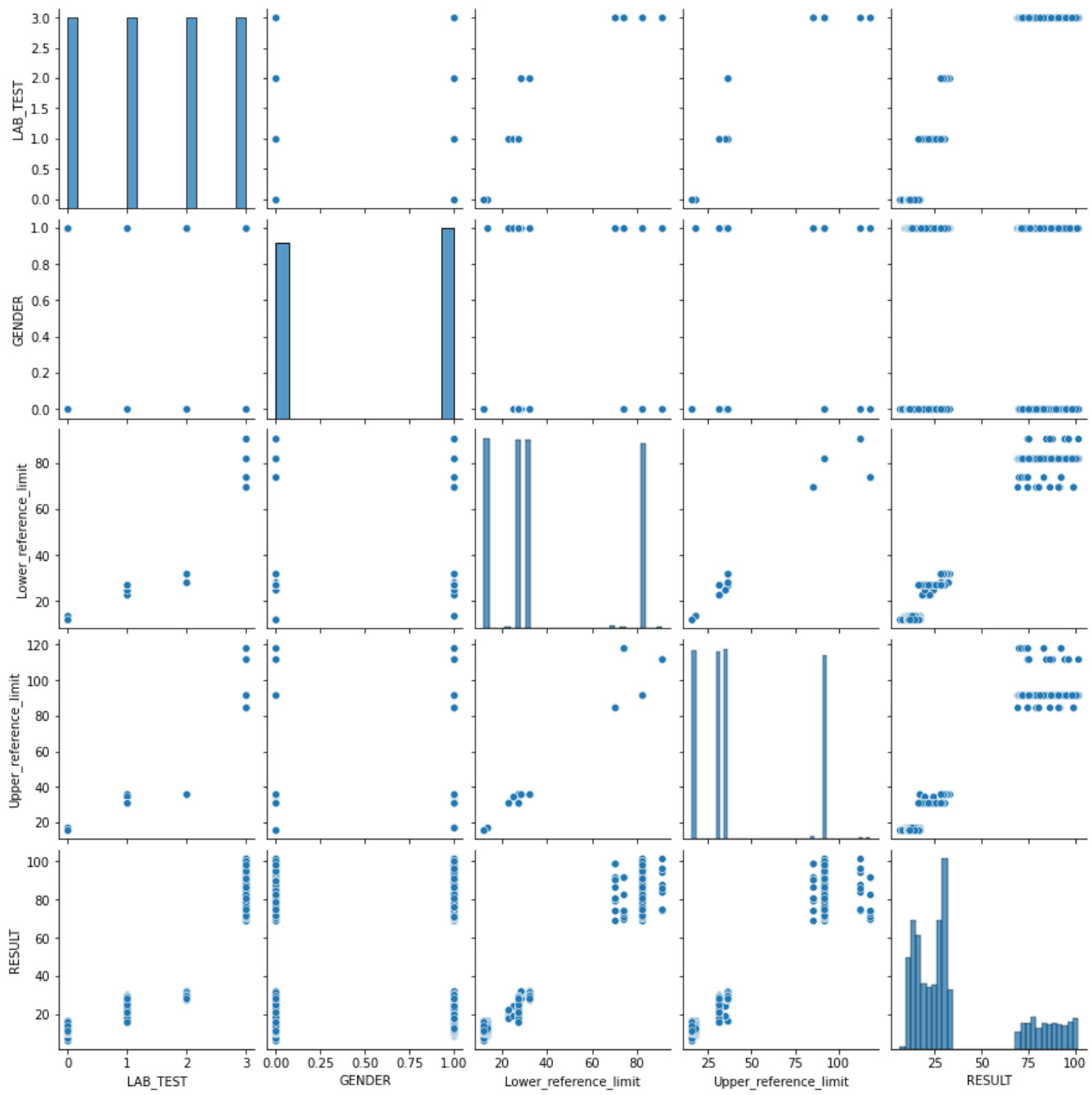
In [22]:
```
data.columns
```

Out[22]:
```
Index(['LAB_TEST', 'GENDER', 'Lower_reference_limit', 'Upper_reference_limit',
       'RESULT', 'IDENTIFICATION'],
      dtype='object')
```

In [23]:
```
data['Upper_reference_limit'] =
data['Upper_reference_limit'].astype(float)
data['Lower_reference_limit'] =
data['Lower_reference_limit'].astype(float)
```

In [24]:
```
pairplot = data[['LAB_TEST', 'GENDER', 'Lower_reference_limit',
'Upper_reference_limit',
       'RESULT']]
sns.pairplot(pairplot)
```

Out[24]:
```
<seaborn.axisgrid.PairGrid at 0x255b0675df0>
```



In [25]:
```
#plt.scatter(range(0,5684),data['RESULT'])
#sns.boxplot(data['RESULT'])
```

In [26]:
```
x = data.drop(columns = 'IDENTIFICATION')
y = data['IDENTIFICATION']
```

4) Scaling the Data Set

In [27]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_x = scaler.fit_transform(x)
```

In [28]:
```
numerical_columns =
['LAB_TEST','GENDER','Lower_reference_limit','Upper_reference_limit']
```

In [29]:
```
data[numerical_columns].info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 5684 entries, 0 to 5683
Data columns (total 4 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   LAB_TEST               5684 non-null   int32
 1   GENDER                 5684 non-null   int32
 2   Lower_reference_limit  5684 non-null   float64
 3   Upper_reference_limit  5684 non-null   float64
dtypes: float64(2), int32(2)
memory usage: 133.3 KB
```

In [30]:

```
data['LAB_TEST'].astype('int64')
data['GENDER'].astype('int64')
data[numerical_columns].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5684 entries, 0 to 5683
Data columns (total 4 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   LAB_TEST               5684 non-null   int32
 1   GENDER                 5684 non-null   int32
 2   Lower_reference_limit  5684 non-null   float64
 3   Upper_reference_limit  5684 non-null   float64
dtypes: float64(2), int32(2)
memory usage: 133.3 KB
```

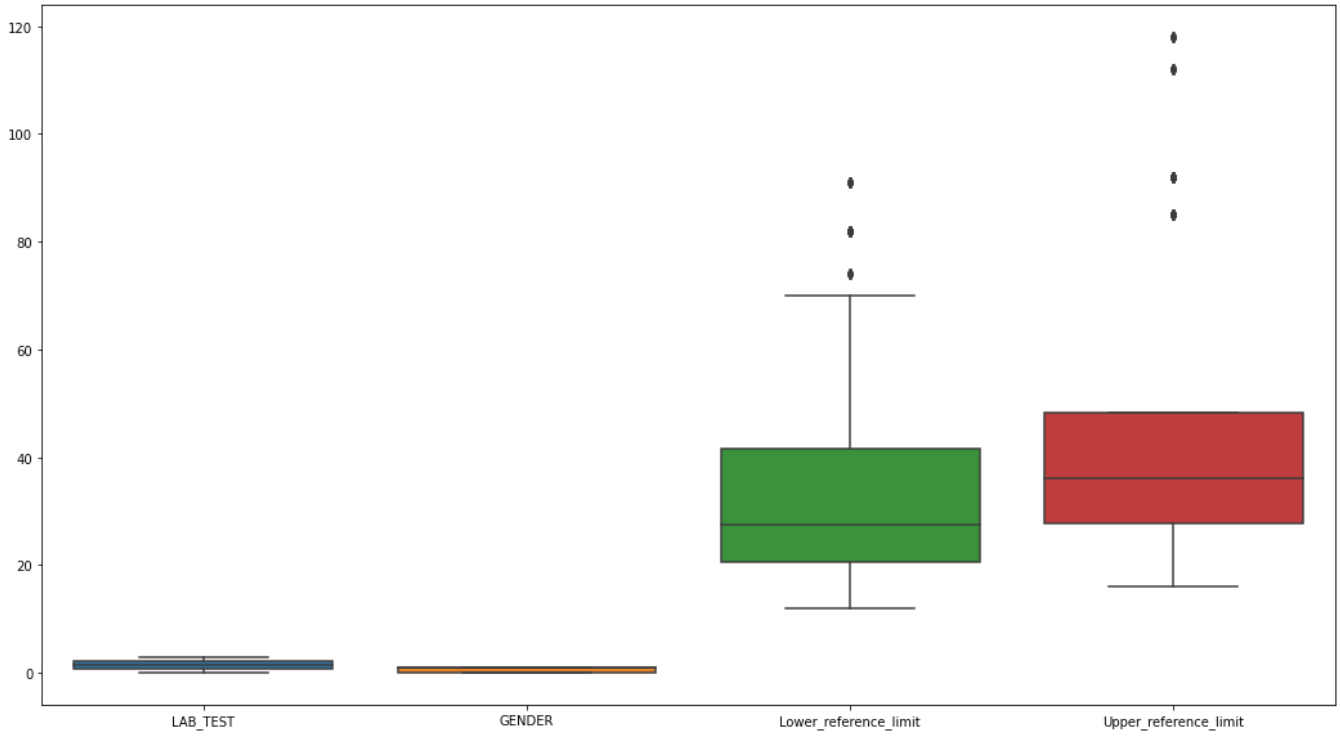
5) Treating the outliers

In [31]:

```
plt.figure(figsize = (18,10))
sns.boxplot(data=data[numerical_columns])
```

Out[31]:

```
<AxesSubplot:>
```



In [32]:

```
def cap_data(data):
    for col in numerical_columns:
        if (((data[col].dtype)=='float64') |
((data[col].dtype)=='int64')):
            percentiles = data[col].quantile([0.25,0.75]).values
            iqr = percentiles[1]-percentiles[0]
            upper_limit = percentiles[1] + 1.5*iqr
```

```
            lower_limit = percentiles[0] -1.5*iqr
            data[col][data[col] <= percentiles[0]] = lower_limit
            data[col][data[col] >= percentiles[1]] = upper_limit
        else:
            data[col]=data[col]
    return data

cap_data(data)
```

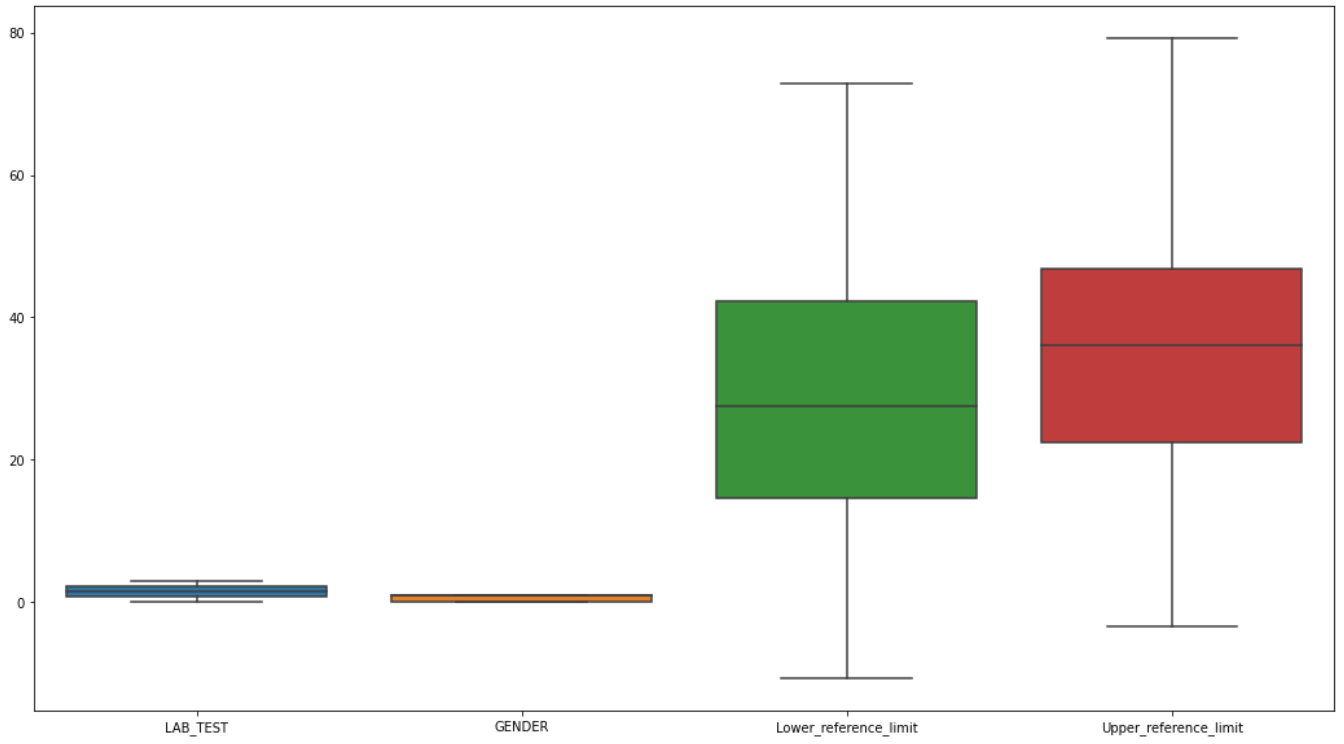
Out[32]:

	LAB_TEST	GENDER	Lower_reference_limit	Upper_reference_limit	RESULT	IDENTIFICATION
0	0	1	-10.6875	-3.3125	14.9	0
1	1	1	27.0000	31.0000	22.7	0
2	2	1	32.0000	36.0000	29.1	0
3	3	1	72.8125	79.1875	83.7	0
4	0	0	-10.6875	-3.3125	15.9	0
...	...	...	...	...	...	...
5679	3	0	72.8125	79.1875	95.2	0
5680	0	0	-10.6875	-3.3125	11.8	1
5681	1	0	27.0000	31.0000	21.2	1
5682	2	0	32.0000	36.0000	28.4	1
5683	3	0	72.8125	79.1875	98.1	1

5684 rows × 6 columns

In [33]:
```
plt.figure(figsize = (18,10))
sns.boxplot(data=data[numerical_columns])
```

Out[33]:
```
<AxesSubplot:>
```



6) Treating the Multicollinearity

In [34]:

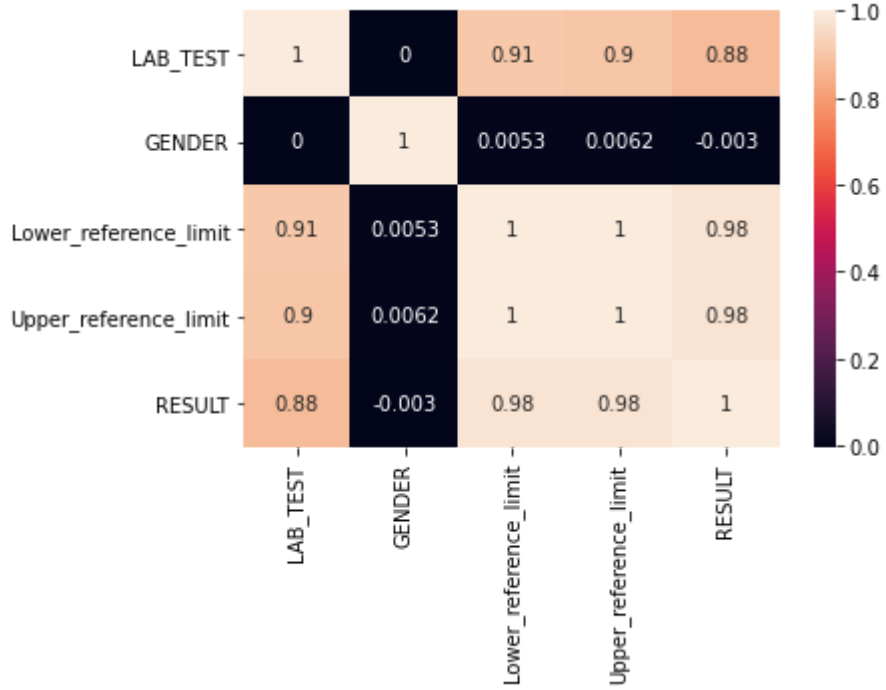
```
corr_matrix = x.corr()
corr_matrix.head()
```

Out[34]:

	LAB_TEST	GENDER	Lower_reference_limit	Upper_reference_limit	RESULT
LAB_TEST	1.000000	0.000000	0.909774	0.899174	0.875279
GENDER	0.000000	1.000000	0.005289	0.006153	-0.003005
Lower_reference_limit	0.909774	0.005289	1.000000	0.998112	0.979040
Upper_reference_limit	0.899174	0.006153	0.998112	1.000000	0.979095
RESULT	0.875279	-0.003005	0.979040	0.979095	1.000000

In [35]:

```
sns.heatmap(corr_matrix,annot = True)
plt.show()
```



Splitting the dataset

In [36]:

```
from sklearn.model_selection import train_test_split as tts
x_train, x_test, y_train,y_test = tts(scaled_x, y, train_size =
0.7,stratify = None)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[36]:

```
((3978, 5), (1706, 5), (3978,), (1706,))
```

Classification Model

1) Logistic Regression

In [37]:

```
from sklearn.linear_model import LogisticRegression as LR
classifier = LR(class_weight = 'balanced')
```

In [38]:

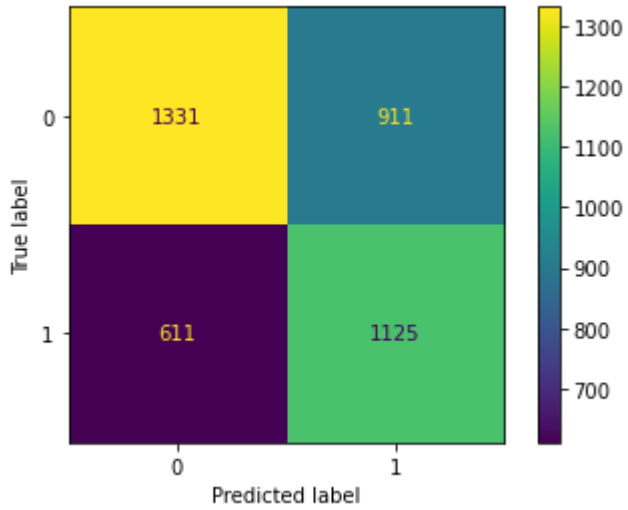
```
classifier.fit(x_train,y_train)
```

Out[38]:
```
LogisticRegression(class_weight='balanced')
```

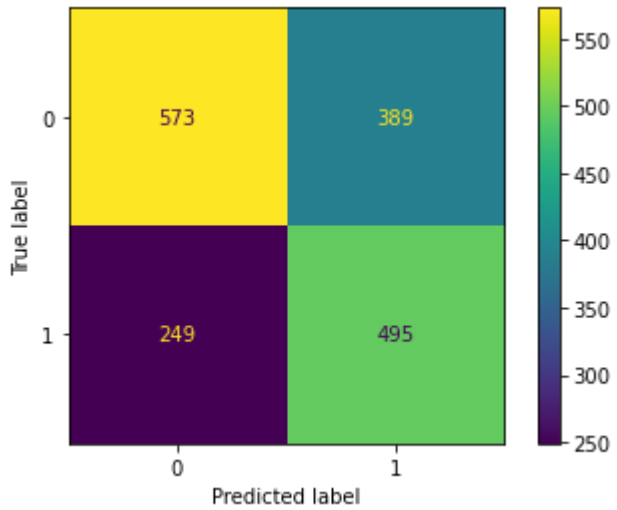
In [39]:
```
y_pred = classifier.predict(x_test)
predicted_proba = classifier.predict_proba(x_test)
predicted_proba
```

Out[39]:
```
array([[0.39110252, 0.60889748],
       [0.55191901, 0.44808099],
       [0.38866184, 0.61133816],
       ...,
       [0.41320358, 0.58679642],
       [0.60216087, 0.39783913],
       [0.38866184, 0.61133816]])
```

In [40]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, x_train,y_train)
plt.show()
```



In [41]:
```
plot_confusion_matrix(classifier, x_test,y_test)
plt.show()
```



In [42]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

Out[42]:
```
0.6260257913247362
```

In [43]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.70      0.60      0.64       962
           1       0.56      0.67      0.61       744

    accuracy                           0.63      1706
   macro avg       0.63      0.63      0.63      1706
weighted avg       0.64      0.63      0.63      1706
```

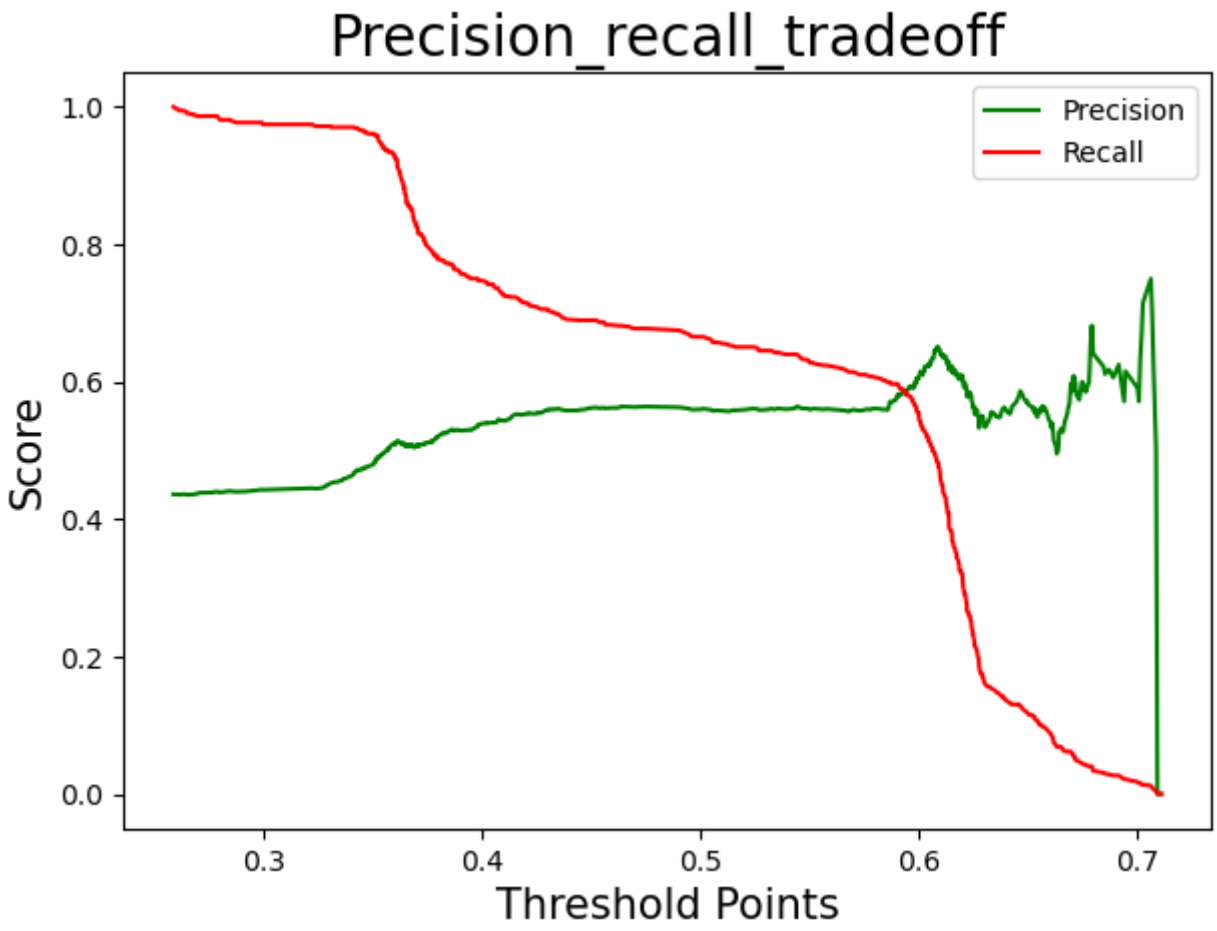
In [44]:
```
from sklearn.metrics import precision_recall_curve
precision_points,recall_points,threshold_points =
precision_recall_curve(y_test,

predicted_proba[:,1],pos_label = '1')
precision_points.shape,recall_points.shape,threshold_points.shape
```

Out[44]:
```
((678,), (678,), (677,))
```

In [45]:
```
plt.figure(figsize = (7,5),dpi = 100)
plt.plot(threshold_points,precision_points[:-1],color = 'green',label =
'Precision')
plt.plot(threshold_points, recall_points[:-1],color = 'red',label =
'Recall')
plt.xlabel('Threshold Points', fontsize = 15)
plt.ylabel('Score',fontsize = 15)
plt.title('Precision_recall_tradeoff',fontsize = 20)
plt.legend()
```

Out[45]:
```
<matplotlib.legend.Legend at 0x255b31edf10>
```

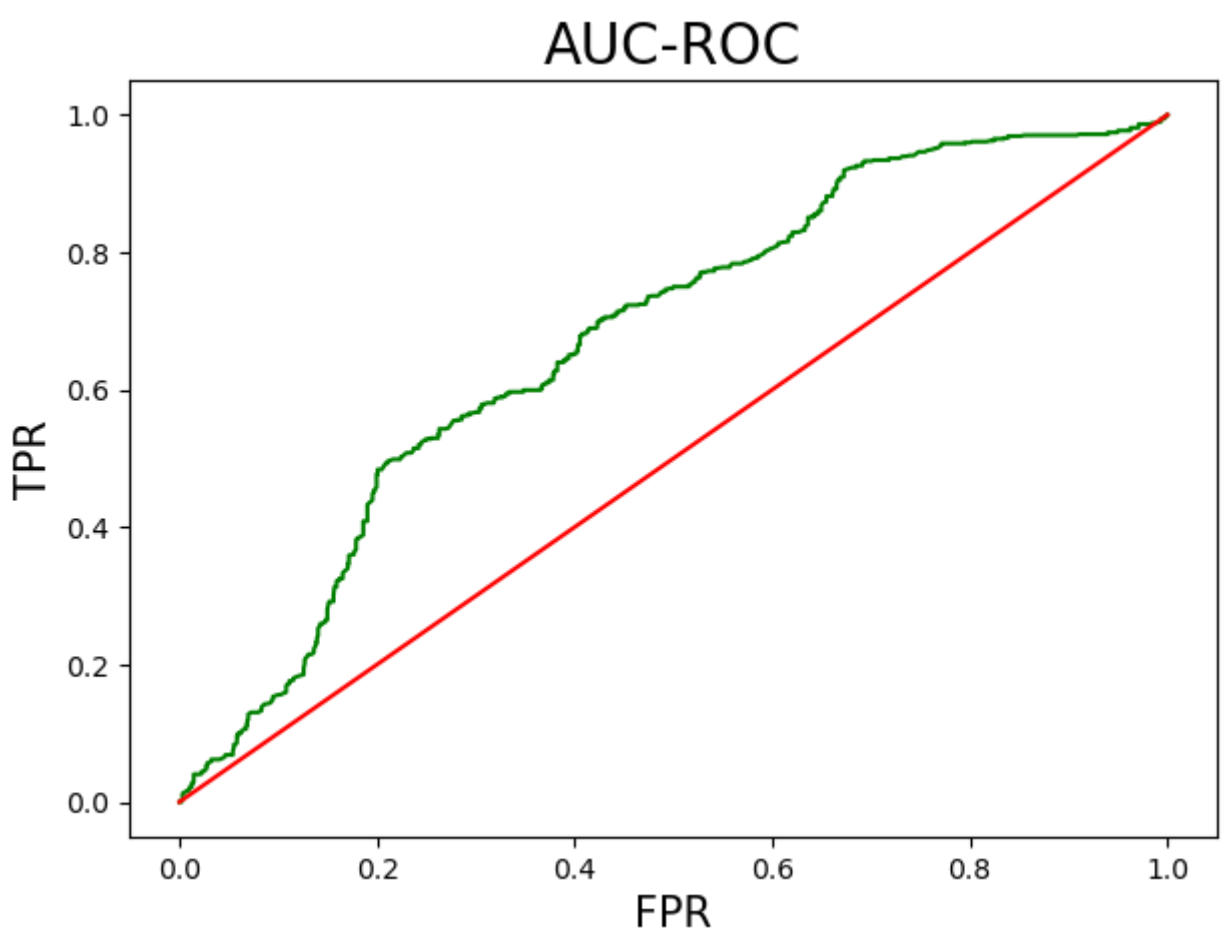


In [46]:

```
from sklearn.metrics import roc_curve, roc_auc_score
fpr,tpr,threshold = roc_curve(y_test,predicted_proba[:,-1],pos_label =
'1')
```

In [47]:

```
plt.figure(figsize = (7,5),dpi = 100)
plt.plot(fpr,tpr, color = 'green')
plt.plot([0,1],[0,1],label = 'baseline', color = 'red')
plt.xlabel('FPR', fontsize = 15)
plt.ylabel('TPR', fontsize = 15)
plt.title('AUC-ROC', fontsize = 20)
plt.show()
roc_auc_score(y_test,predicted_proba[:,1])
```



Out[47]:
```
0.6748359712069388
```

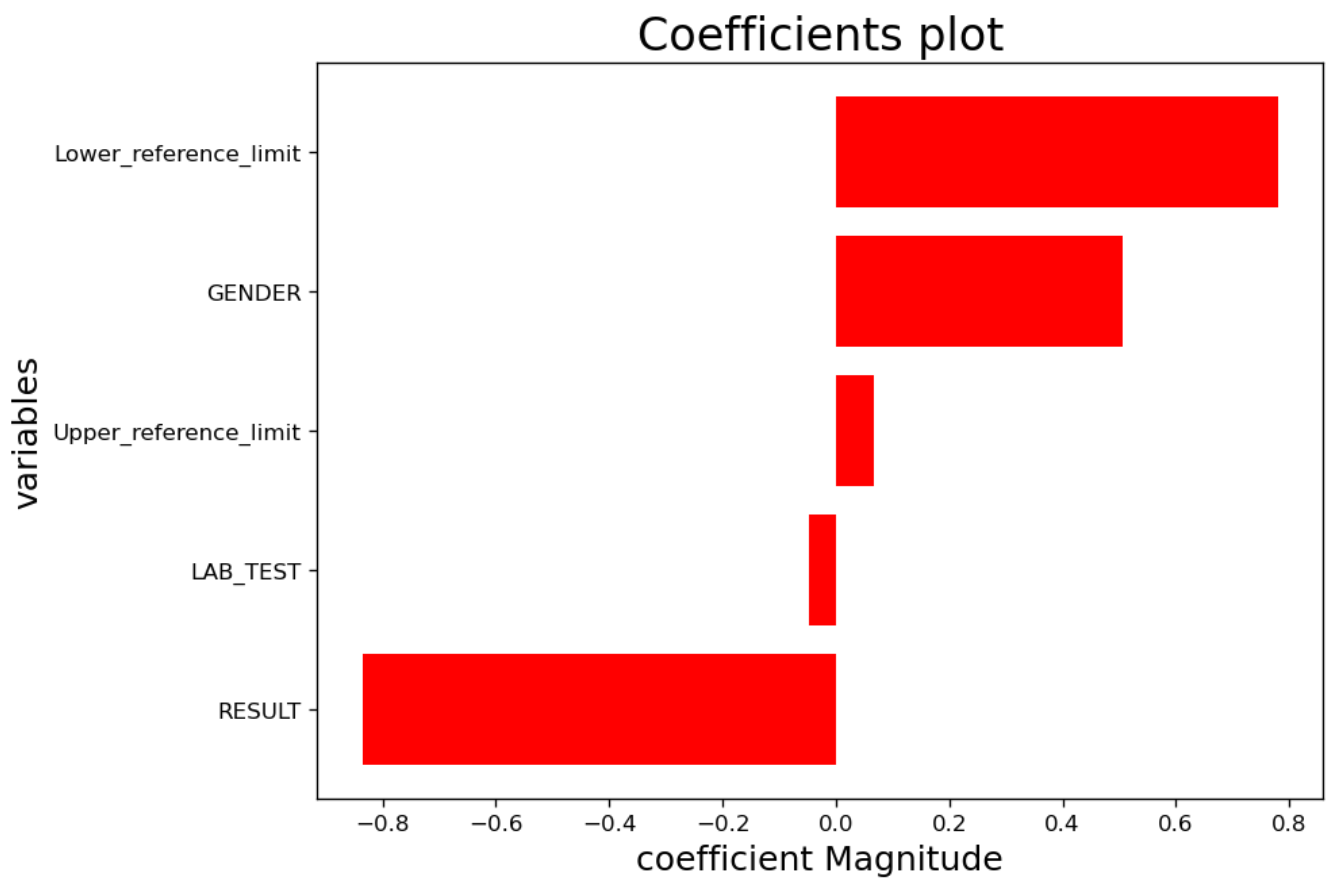
In [48]:
```
c = classifier.coef_.reshape(-1)
x = x.columns
coeff_plot = pd.DataFrame({
    'coefficients':c,
    'variables':x,
    })
#sorting the values
coeff_plot = coeff_plot.sort_values(by = 'coefficients')
coeff_plot.head()
```

Out[48]:

	coefficients	variables
4	-0.834313	RESULT
0	-0.047015	LAB_TEST
3	0.067038	Upper_reference_limit
1	0.506983	GENDER
2	0.779267	Lower_reference_limit

In [59]:
```
plt.figure(figsize = (8,6),dpi = 120)
plt.barh(coeff_plot['variables'],coeff_plot['coefficients'],color =
'red')
plt.xlabel('coefficient Magnitude',fontsize = 15)
plt.ylabel('variables',fontsize = 15)
plt.title('Coefficients plot',fontsize = 20)
```

Out[59]:
```
Text(0.5, 1.0, 'Coefficients plot')
```



2) Random Forest

In [50]:
```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=300)
rfc.fit(x_train, y_train)
y_pred2 = rfc.predict(x_test)
```

In [51]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test,y_pred2)
accuracy
```

Out[51]:
```
0.8106682297772567
```

In [52]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred2)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83       962
           1       0.78      0.80      0.79       744

    accuracy                           0.81      1706
   macro avg       0.81      0.81      0.81      1706
weighted avg       0.81      0.81      0.81      1706
```

3) K- Nearest Neighbours

In [53]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
```

```
knn.fit(x_train, y_train)

y_pred3 = knn.predict(x_test)
```

In [54]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test,y_pred3)
accuracy
```

Out[54]:
```
0.7467760844079718
```

In [55]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred3)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.77      0.79      0.78       962
           1       0.72      0.69      0.70       744

    accuracy                           0.75      1706
   macro avg       0.74      0.74      0.74      1706
weighted avg       0.75      0.75      0.75      1706
```

4) Support Vector Machine

In [56]:
```
from sklearn.svm import SVC

svc_model = SVC(kernel="rbf")
svc_model.fit(x_train, y_train)
y_pred4 = svc_model.predict(x_test)
```

In [57]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test,y_pred4)
accuracy
```

Out[57]:
```
0.6518171160609613
```

In [58]:
```
from sklearn.metrics import classification_report
k = classification_report(y_test,y_pred4)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.72      0.63      0.67       962
           1       0.59      0.68      0.63       744

    accuracy                           0.65      1706
   macro avg       0.65      0.65      0.65      1706
weighted avg       0.66      0.65      0.65      1706
```

On comparing all these models it is found that the Random Forest model predicts most precisely with a recall score of (0.82,0.8)

CHAPTER 6:
MACHINE LEARNING MODELS

DIABETES PREDICTION

Diabetes Prediction

1) Importing the necessary python libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
pd.options.mode.chained_assignment = None
```

2) Importing the dataset

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

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Out
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

In [3]:
```
data.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```
data.columns
```

Out[4]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Exploratory Data Analysis

3) Data Cleaning

In [6]:
```
data.columns.isnull().sum()
```

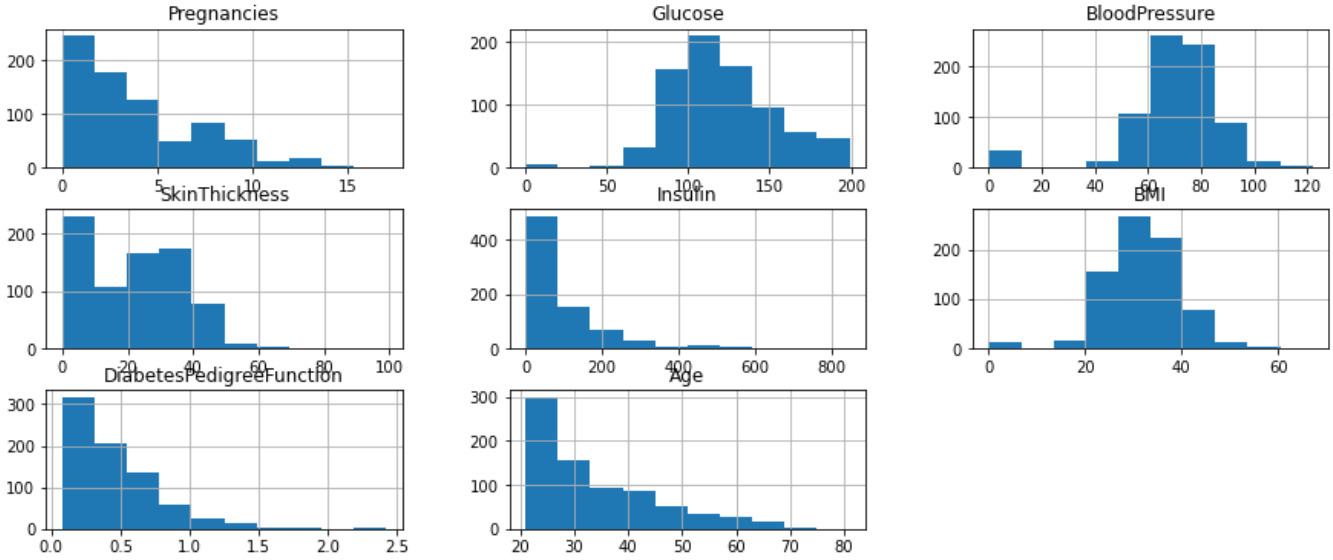
Out[6]:
```
0
```

In [7]:
```
data['Outcome'].value_counts()/len(data)
```

Out[7]:
```
0    0.651042
1    0.348958
Name: Outcome, dtype: float64
```

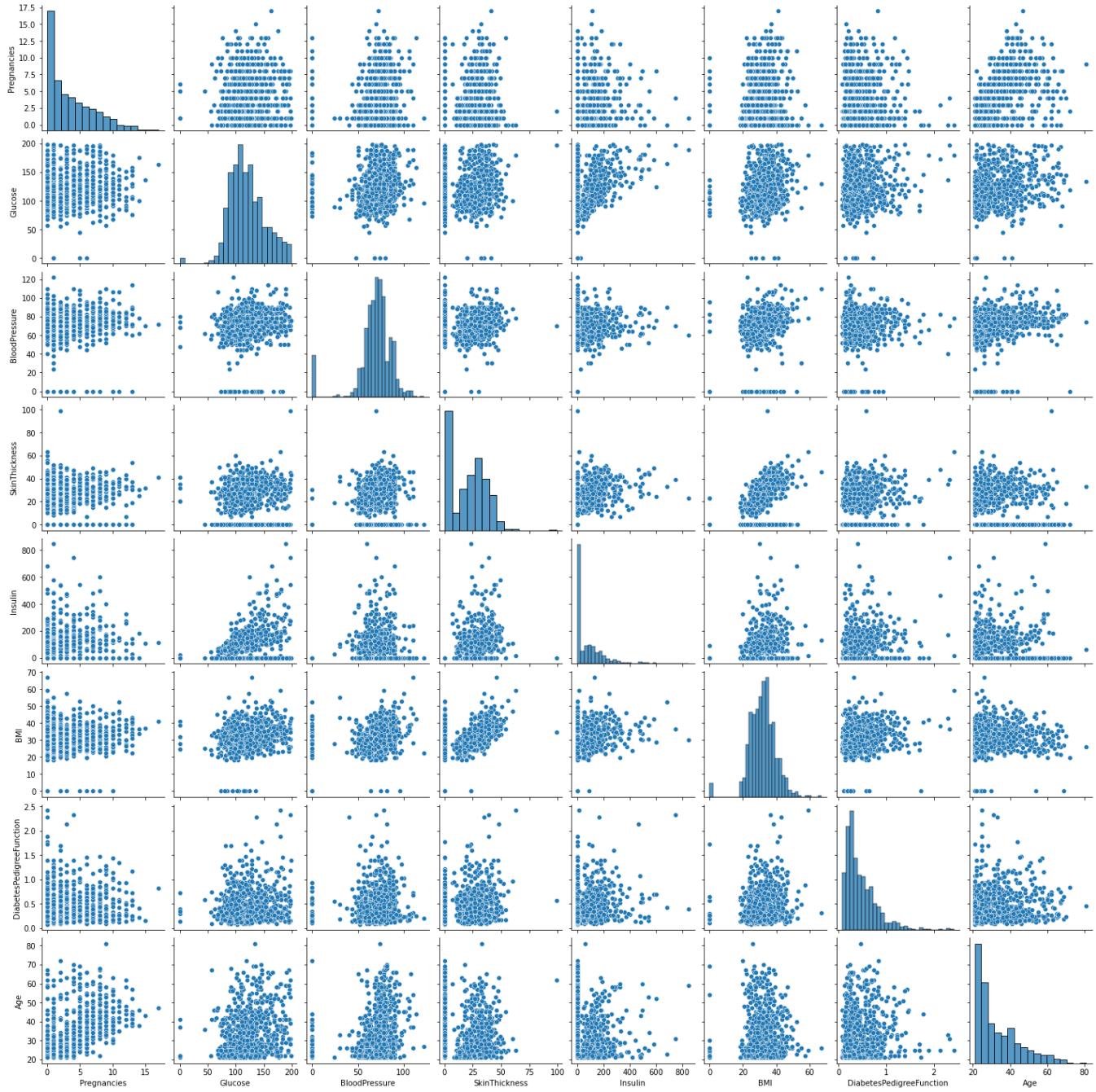
In [8]:
```
Numerical_columns = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']
```

In [9]:
```
data[Numerical_columns].hist(bins = 10, figsize = (15,6));
```



In [10]:
```
pairplot = data[Numerical_columns]
sns.pairplot(pairplot)
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x1a8c9a3bdc0>
```



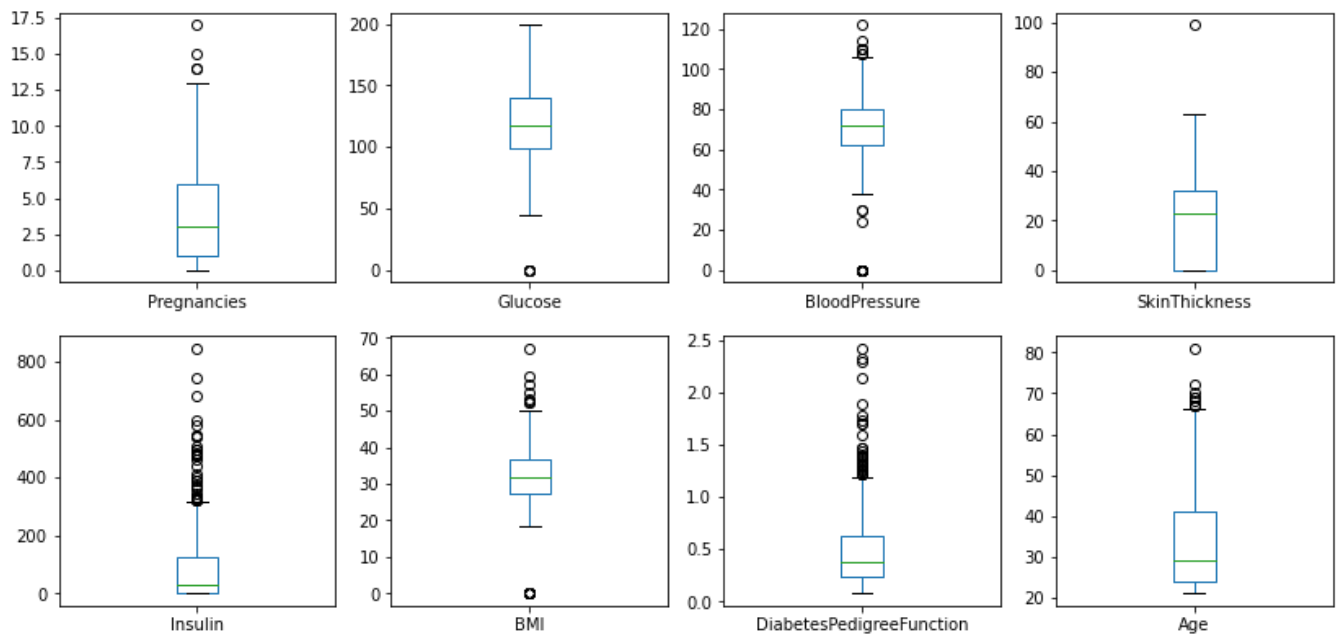
4) Treating the Outliers

In [11]:

```
data[Numerical_columns].plot(kind = 'box', subplots = 'True',
                             layout = (4,4),fontsize = 10,figsize =
(14,14))
```

Out[11]:

```
Pregnancies                 AxesSubplot(0.125,0.71587;0.168478x0.16413)
Glucose                   AxesSubplot(0.327174,0.71587;0.168478x0.16413)
BloodPressure             AxesSubplot(0.529348,0.71587;0.168478x0.16413)
SkinThickness             AxesSubplot(0.731522,0.71587;0.168478x0.16413)
Insulin                     AxesSubplot(0.125,0.518913;0.168478x0.16413)
BMI                       AxesSubplot(0.327174,0.518913;0.168478x0.16413)
DiabetesPedigreeFunction  AxesSubplot(0.529348,0.518913;0.168478x0.16413)
Age                       AxesSubplot(0.731522,0.518913;0.168478x0.16413)
dtype: object
```



In [12]:

```
def cap_data(data):
    for col in Numerical_columns:
        if (((data[col].dtype)=='float64') |
((data[col].dtype)=='int64')):
            percentiles = data[col].quantile([0.25,0.75]).values
            iqr = percentiles[1]-percentiles[0]
            upper_limit = percentiles[1] + 1.5*iqr
            lower_limit = percentiles[0] -1.5*iqr
            data[col][data[col] <= percentiles[0]] = lower_limit
            data[col][data[col] >= percentiles[1]] = upper_limit
        else:
            data[col]=data[col]
    return data

cap_data(data)
```

Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	13.5	202.125	72	80	-190.875	33.60	1.200	66.5
1	-6.5	37.125	66	29	-190.875	13.35	0.351	31.0
2	13.5	202.125	64	-48	-190.875	13.35	1.200	32.0
3	-6.5	37.125	66	23	94.000	28.10	-0.330	-1.5
4	-6.5	137.000	35	80	318.125	50.55	1.200	33.0
...	...	...	...	...	...	...	...	...
763	13.5	101.000	76	80	318.125	32.90	-0.330	66.5
764	2.0	122.000	70	27	-190.875	50.55	0.340	27.0
765	5.0	121.000	72	23	112.000	13.35	0.245	30.0
766	-6.5	126.000	35	-48	-190.875	30.10	0.349	66.5
767	-6.5	37.125	70	31	-190.875	30.40	0.315	-1.5

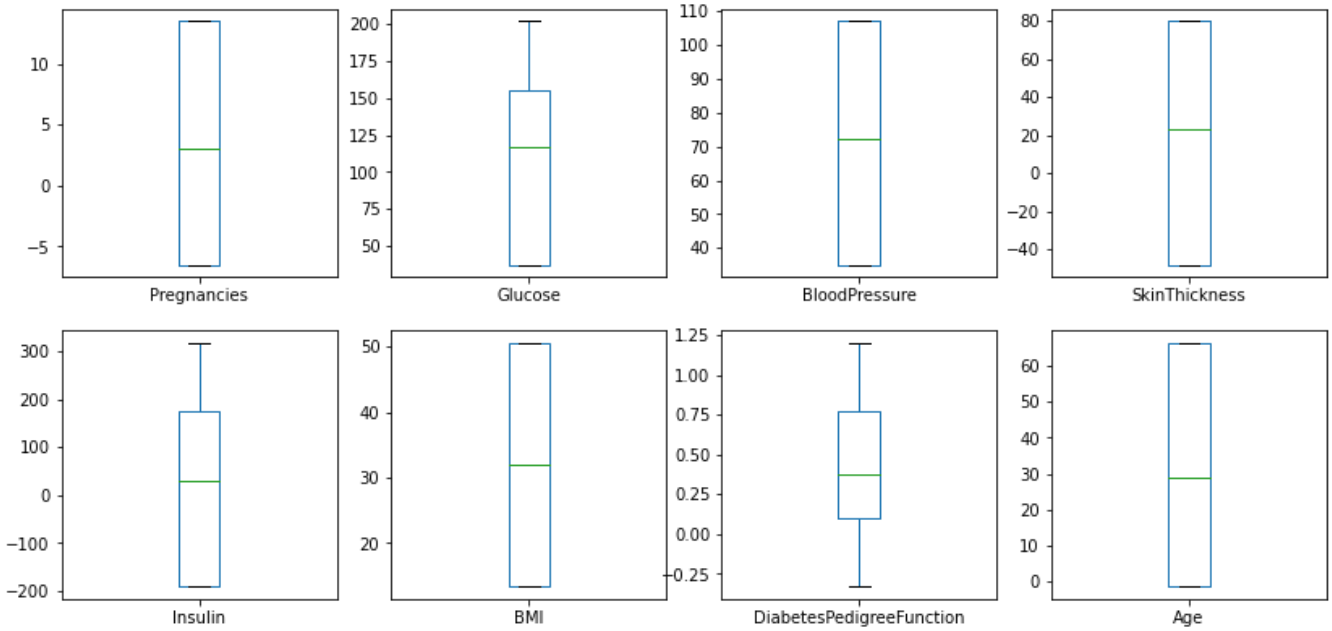
768 rows × 9 columns

In [13]:

```
data[Numerical_columns].plot(kind = 'box', subplots = 'True',
                             layout = (4,4),fontsize = 10,figsize =
(14,14))
```

Out[13]:

```
Pregnancies                 AxesSubplot(0.125,0.71587;0.168478x0.16413)
Glucose                   AxesSubplot(0.327174,0.71587;0.168478x0.16413)
BloodPressure             AxesSubplot(0.529348,0.71587;0.168478x0.16413)
SkinThickness             AxesSubplot(0.731522,0.71587;0.168478x0.16413)
Insulin                     AxesSubplot(0.125,0.518913;0.168478x0.16413)
BMI                       AxesSubplot(0.327174,0.518913;0.168478x0.16413)
DiabetesPedigreeFunction  AxesSubplot(0.529348,0.518913;0.168478x0.16413)
Age                       AxesSubplot(0.731522,0.518913;0.168478x0.16413)
dtype: object
```



In [14]:

```
data.describe()
```

Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768
mean	3.054036	118.144043	71.636719	18.010417	7.153320	31.915299	0
std	7.806577	59.300407	26.109978	49.095016	211.833256	13.336844	0
min	-6.500000	37.125000	35.000000	-48.000000	-190.875000	13.350000	-0
25%	-6.500000	37.125000	35.000000	-48.000000	-190.875000	13.350000	0
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0
75%	13.500000	155.531250	107.000000	80.000000	174.781250	50.550000	0
max	13.500000	202.125000	107.000000	80.000000	318.125000	50.550000	1

In [15]:

```
X = data.drop(columns = ['Outcome'])
Y = data['Outcome']
```

5) Scaling the Dataset

In [16]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_X = scaler.fit_transform(X)
```

6) Treating Multicollinearity

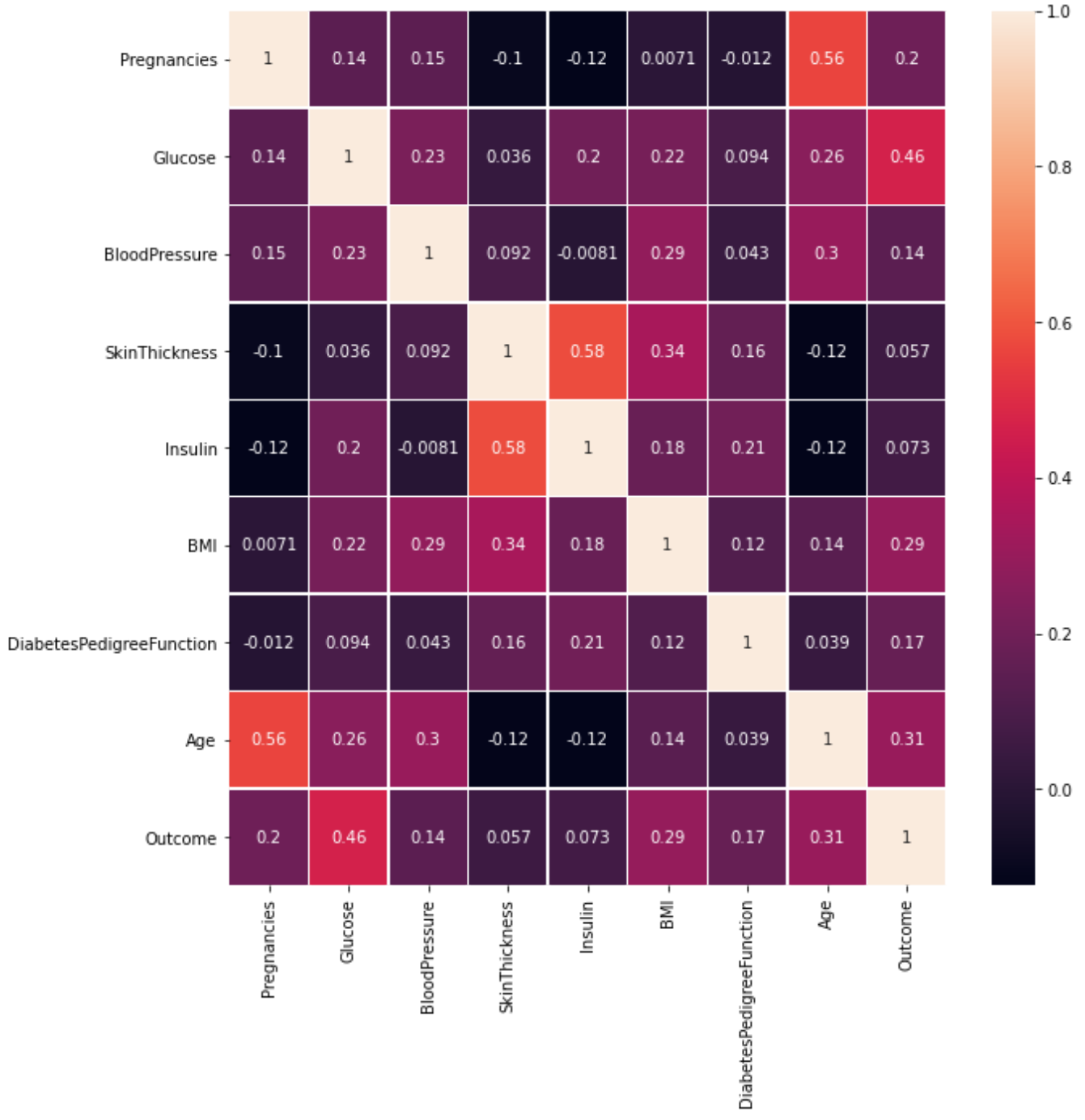
In [17]:
```
corr_matrix = X.corr()
corr_matrix.head()
```

Out[17]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
Pregnancies	1.000000	0.142069	0.145615	-0.100834	-0.123861	0.007127	
Glucose	0.142069	1.000000	0.227260	0.035919	0.197412	0.216758	
BloodPressure	0.145615	0.227260	1.000000	0.091779	-0.008057	0.286807	
SkinThickness	-0.100834	0.035919	0.091779	1.000000	0.579186	0.343473	
Insulin	-0.123861	0.197412	-0.008057	0.579186	1.000000	0.175972	

In [18]:
```
plt.figure(figsize=(10,10))
sns.heatmap(data.corr(), annot = True, linewidths = .5)
```

Out[18]:
```
<AxesSubplot:>
```



Splitting the Dataset

In [19]:
```
from sklearn.model_selection import train_test_split as tts
X_train,X_test,Y_train, Y_test = tts(scaled_X, Y, train_size =
0.7,stratify = None)
X_train.shape,X_test.shape,Y_train.shape,Y_test.shape
```

Out[19]:
```
((537, 8), (231, 8), (537,), (231,))
```

Classification Models

1) Logistic Regression

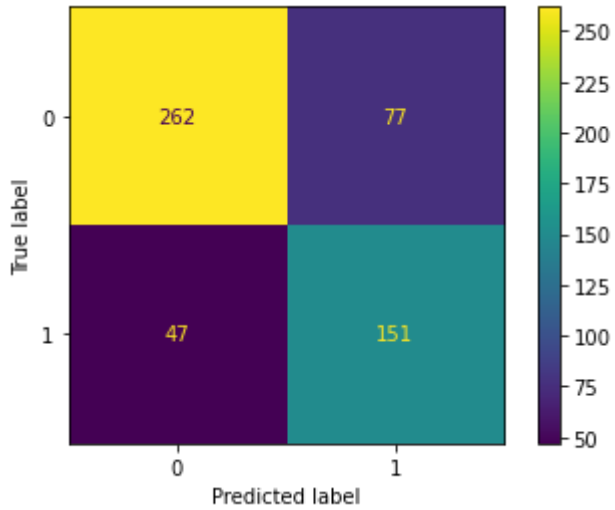
In [20]:
```
from sklearn.linear_model import LogisticRegression as LR
classifier = LR(class_weight = 'balanced')
```

In [21]:
```
classifier.fit(X_train, Y_train)
```

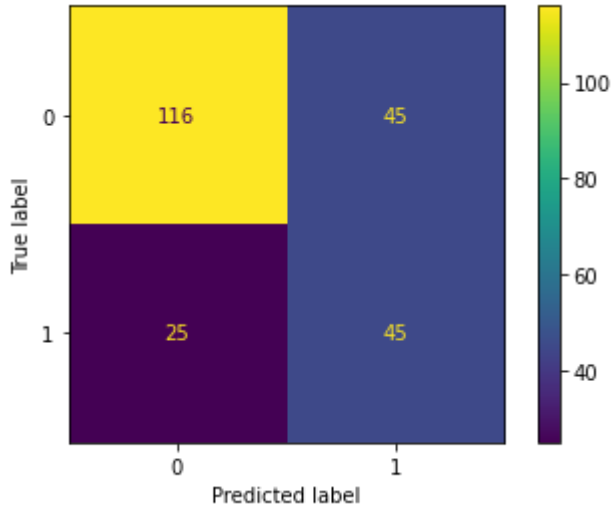
Out[21]:
```
LogisticRegression(class_weight='balanced')
```

In [22]:
```
predicted_values = classifier.predict(X_test)
predicted_probabilities = classifier.predict_proba(X_test)
```

In [23]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_train, Y_train)
plt.show()
```



In [24]:
```
plot_confusion_matrix(classifier,X_test,Y_test)
plt.show()
```



In [25]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test,predicted_values)
accuracy
```

Out[25]:
```
0.696969696969697
```

In [26]:
```
from sklearn.metrics import classification_report
k = classification_report(Y_test,predicted_values)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.82      0.72      0.77       161
           1       0.50      0.64      0.56        70

    accuracy                           0.70       231
   macro avg       0.66      0.68      0.67       231
weighted avg       0.72      0.70      0.71       231
```

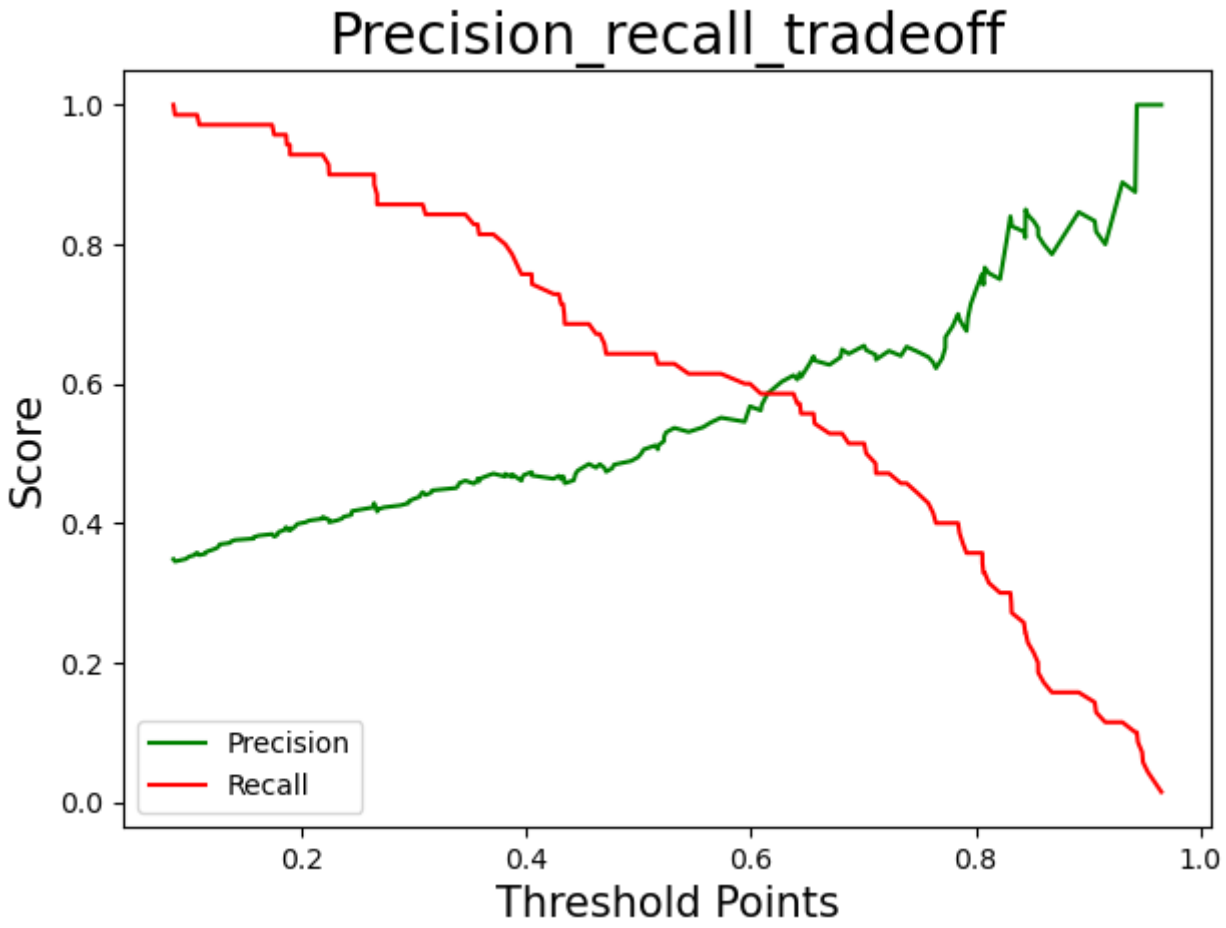
In [27]:
```
from sklearn.metrics import precision_recall_curve
precision_points,recall_points,threshold_points =
precision_recall_curve(Y_test,

predicted_probabilities[:,1])
precision_points.shape,recall_points.shape,threshold_points.shape
```

Out[27]:
```
((202,), (202,), (201,))
```

In [28]:
```
plt.figure(figsize = (7,5),dpi = 100)
plt.plot(threshold_points,precision_points[:-1],color = 'green',label =
'Precision')
plt.plot(threshold_points, recall_points[:-1],color = 'red',label =
'Recall')
plt.xlabel('Threshold Points', fontsize = 15)
plt.ylabel('Score',fontsize = 15)
plt.title('Precision_recall_tradeoff',fontsize = 20)
plt.legend()
```

Out[28]:
```
<matplotlib.legend.Legend at 0x1a8cebf97c0>
```

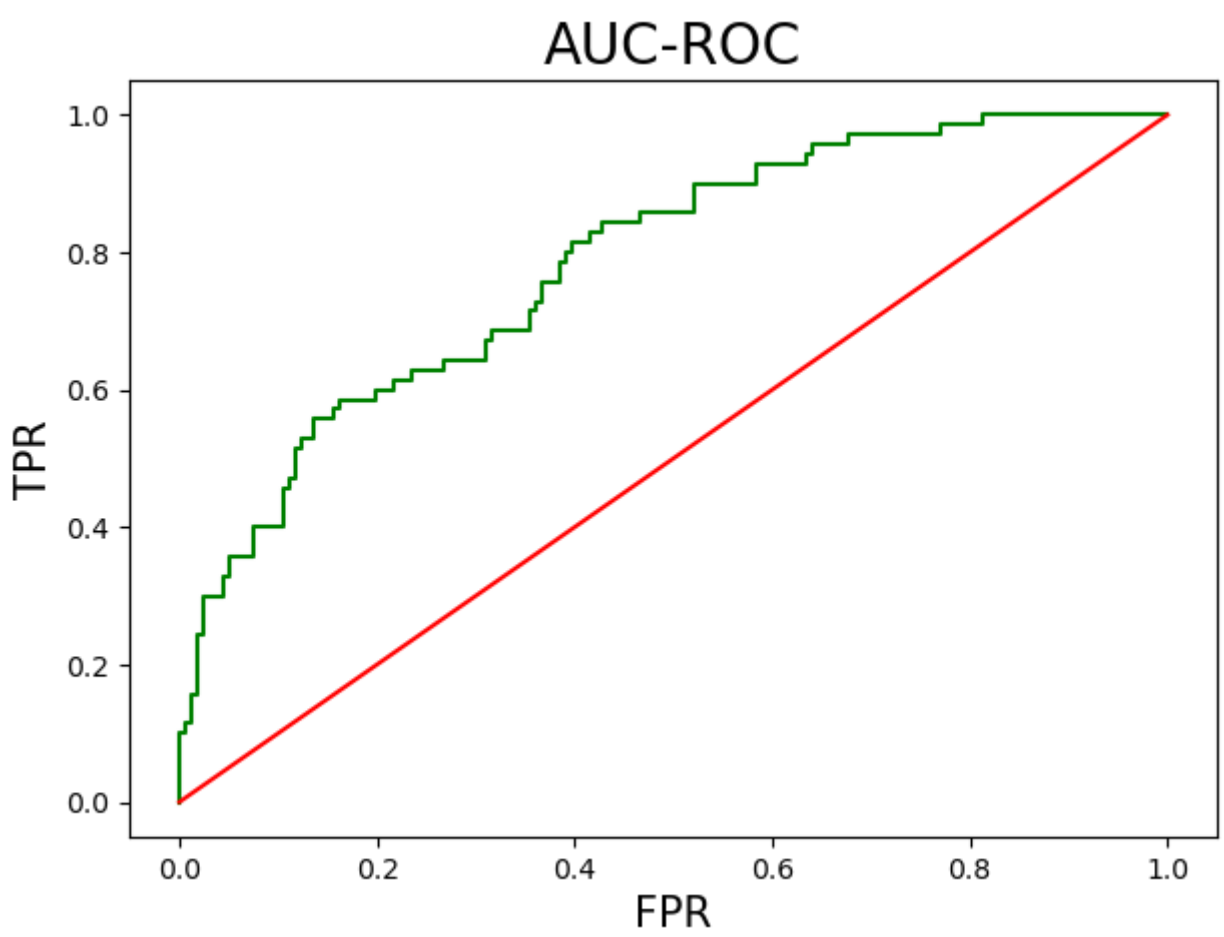


In [29]:

```
#aoc_roc_curve
from sklearn.metrics import roc_curve, roc_auc_score
fpr,tpr,threshold = roc_curve(Y_test,predicted_probabilities[:,-1])
```

In [30]:

```
plt.figure(figsize = (7,5),dpi = 100)
plt.plot(fpr,tpr, color = 'green')
plt.plot([0,1],[0,1],label = 'baseline', color = 'red')
plt.xlabel('FPR', fontsize = 15)
plt.ylabel('TPR', fontsize = 15)
plt.title('AUC-ROC', fontsize = 20)
plt.show()
roc_auc_score(Y_test,predicted_probabilities[:,1])
```



Out[30]:
```
0.784826974267968
```

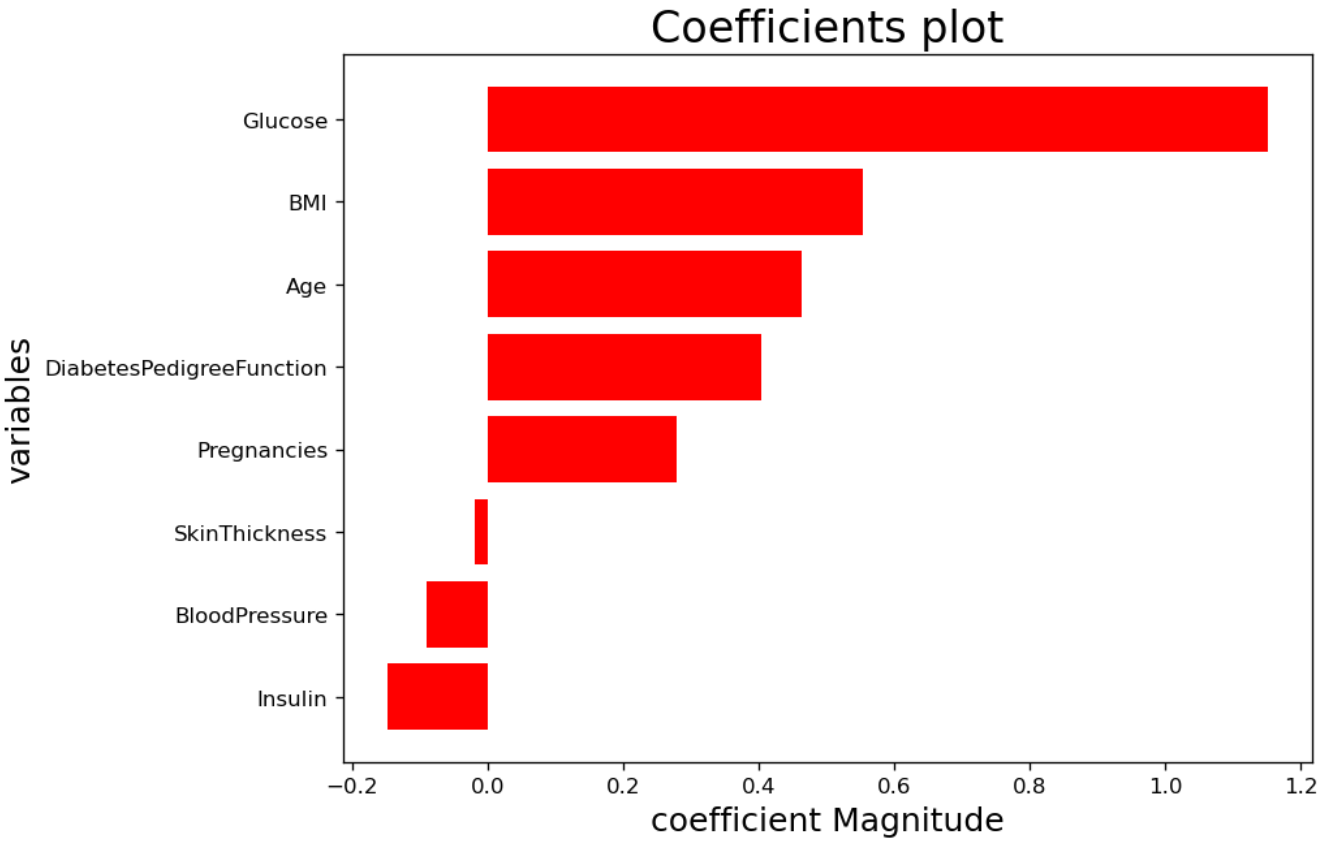
In [31]:
```
#coefficient plot
c = classifier.coef_.reshape(-1)
x = X.columns
coeff_plot = pd.DataFrame({
    'coefficients':c,
    'variables':x,
    })
#sorting the values
coeff_plot = coeff_plot.sort_values(by = 'coefficients')
coeff_plot.head()
```

Out[31]:

	coefficients	variables
4	-0.148490	Insulin
2	-0.091168	BloodPressure
3	-0.019928	SkinThickness
0	0.278092	Pregnancies
6	0.403584	DiabetesPedigreeFunction

In [32]:
```
plt.figure(figsize = (8,6),dpi = 120)
plt.barh(coeff_plot['variables'],coeff_plot['coefficients'],color =
'red')
plt.xlabel('coefficient Magnitude',fontsize = 15)
plt.ylabel('variables',fontsize = 15)
plt.title('Coefficients plot',fontsize = 20)
```

Out[32]:
```
Text(0.5, 1.0, 'Coefficients plot')
```



2) Random Forest

In [33]:
```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=300)
rfc.fit(X_train, Y_train)
y_pred2 = rfc.predict(X_test)
```

In [34]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test,y_pred2)
accuracy
```

Out[34]:
```
0.7402597402597403
```

In [35]:
```
from sklearn.metrics import classification_report
k = classification_report(Y_test,y_pred2)
print(k)
```

```
              precision    recall  f1-score   support

           0       0.82      0.80      0.81       161
           1       0.57      0.60      0.58        70

    accuracy                           0.74       231
   macro avg       0.69      0.70      0.70       231
weighted avg       0.74      0.74      0.74       231
```

In [36]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test,y_pred3)
accuracy
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-36-7695e7e5965f> in <module>
      1 from sklearn.metrics import accuracy_score
----> 2 accuracy = accuracy_score(Y_test,y_pred3)
      3 accuracy

NameError: name 'y_pred3' is not defined
```

3) K-Nearest Neighbours

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, Y_train)

y_pred3 = knn.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test,y_pred4)
accuracy
```

In [ ]:
```
from sklearn.metrics import classification_report
k = classification_report(Y_test,y_pred3)
print(k)
```

4) Support Vector Machine

In [ ]:
```
from sklearn.svm import SVC

svc_model = SVC(kernel="rbf")
svc_model.fit(X_train, Y_train)
y_pred4 = svc_model.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import classification_report
k = classification_report(Y_test,y_pred4)
print(k)
```

On comparing all these models it is found that the support vector machine model predicts most precisely with a recall score of (0.83,0.64)

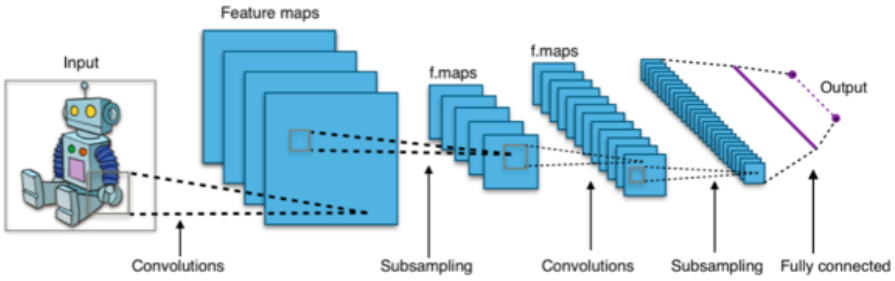
CHAPTER 7:
DEEP LEARNING MODELS

ORAL CANCER PREDICTION USING CONVULTIONAL NEURAL NETWORK(CNN)

CONVULTIONAL NEURAL NETWORK-

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural network, most commonly applied to analyze visual imagery.

For prediction of oral cancer, the dataset used was images of mouth and tongue of cancer patients and non-cancer patients. The CNN was then train on this data to to classify between a non- cancer patient and a cancer patient.



CNN Architecture

Oral Cancer Prediction using Convultion Neural Network(CNN)

Data Processing

1) Importing the neccessary python Libraries

In [1]:
```
!pip install opencv-python
```

```
Collecting opencv-python
  Using cached opencv_python-4.5.2.54-cp38-cp38-win_amd64.whl (34.7 MB)
Requirement already satisfied: numpy>=1.17.3 in c:\programdata\anaconda3\lib\site-pac
kages (from opencv-python) (1.19.2)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.5.2.54
```

In [2]:
```
import numpy as np
import cv2
import os
import random
import matplotlib.pyplot as plt
import pickle
```

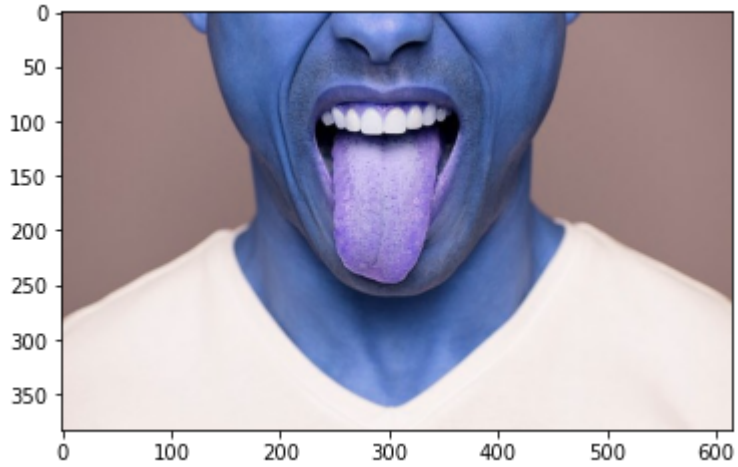
2) Importing the dataset folders containg images of mouth and tongue with oral cancer and without oral cancer

In [3]:
```
DIRECTORY = r"C:\Users\dell\Desktop\machine-learning-
ex\tpcs\OralCancer"
CATEGORIES = ['cancer','non-cancer']
```

In [4]:
```
for category in CATEGORIES:
    folder = os.path.join(DIRECTORY,category)
    for img in os.listdir(folder):
        img_path = os.path.join(folder,img)
        print(img_path)
        break
```

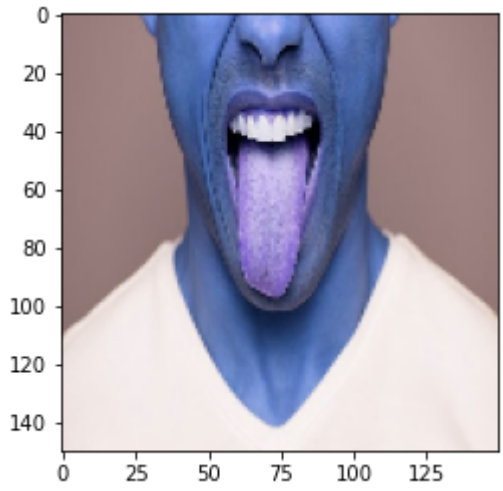
```
C:\Users\dell\Desktop\machine-learning-ex\tpcs\OralCancer\cancer\cancer(1).jpeg
C:\Users\dell\Desktop\machine-learning-ex\tpcs\OralCancer\non-cancer\12654650-6954555
-image-a-22_1556101508834-Edited.jpg
```

In [5]:
```
for category in CATEGORIES:
    folder = os.path.join(DIRECTORY,category)
    for img in os.listdir(folder):
        img_path = os.path.join(folder,img)
        img_array = cv2.imread(img_path)
        plt.imshow(img_array)
        break
```



In [6]:

```
Img_size = 150
for category in CATEGORIES:
    folder = os.path.join(DIRECTORY,category)
    for img in os.listdir(folder):
        img_path = os.path.join(folder,img)
        img_array = cv2.imread(img_path)
        img_array = cv2.resize(img_array,(Img_size,Img_size))
        plt.imshow(img_array)
        break
```



In [7]:

```
Img_size = 150
data = []
for category in CATEGORIES:
    folder = os.path.join(DIRECTORY,category)
    label = CATEGORIES.index(category)
    for img in os.listdir(folder):
        img_path = os.path.join(folder,img)
        img_array = cv2.imread(img_path)
        img_array = cv2.resize(img_array,(Img_size,Img_size))
        data.append([img_array, label])
```

In [8]:

```
len(data)
```

Out[8]:

```
131
```

In [9]:
```
random.shuffle(data)
```

In [10]:
```
data[0]
```

Out[10]:
```
[array([[[149, 179, 218],
         [138, 182, 235],
         [141, 190, 252],
         ...,
         [172, 198, 245],
         [179, 206, 253],
         [197, 209, 241]],

        [[152, 181, 220],
         [143, 187, 240],
         [130, 179, 241],
         ...,
         [176, 200, 241],
         [184, 206, 248],
         [201, 214, 246]],

        [[158, 188, 227],
         [137, 181, 234],
         [127, 176, 239],
         ...,
         [190, 208, 242],
         [196, 212, 246],
         [203, 215, 247]],

        ...,

        [[149, 179, 227],
         [131, 171, 234],
         [129, 169, 234],
         ...,
         [130, 164, 234],
         [130, 164, 234],
         [116, 170, 236]],

        [[154, 184, 231],
         [136, 176, 239],
         [134, 174, 240],
         ...,
         [125, 159, 229],
         [125, 159, 229],
         [124, 176, 236]],

        [[148, 179, 226],
         [142, 182, 245],
         [137, 177, 243],
         ...,
         [135, 169, 239],
         [138, 172, 242],
         [126, 173, 226]]], dtype=uint8),
 0]
```

In [11]:
```
x = []
y = []
for features,labels in data:
    x.append(features)
    y.append(labels)
```

In [12]:
```
x = np.array(x)
y = np.array(y)
```

In [13]:
```

```

```
len(x),len(y)
```

Out[13]:
```
(131, 131)
```

In [14]:
```
pickle.dump(x,open('x.pkl','wb'))
pickle.dump(y,open('y.pkl','wb'))
```

Classification

In [21]:
```
import pickle
import time
```

In [22]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

In [23]:
```
x = pickle.load(open('x.pkl','rb'))
y = pickle.load(open('y.pkl','rb'))
```

In [24]:
```
x
```

Out[24]:
```
array([[[[149, 179, 218],
         [138, 182, 235],
         [141, 190, 252],
         ...,
         [172, 198, 245],
         [179, 206, 253],
         [197, 209, 241]],

        [[152, 181, 220],
         [143, 187, 240],
         [130, 179, 241],
         ...,
         [176, 200, 241],
         [184, 206, 248],
         [201, 214, 246]],

        [[158, 188, 227],
         [137, 181, 234],
         [127, 176, 239],
         ...,
         [190, 208, 242],
         [196, 212, 246],
         [203, 215, 247]],

        ...,

        [[149, 179, 227],
         [131, 171, 234],
         [129, 169, 234],
         ...,
         [130, 164, 234],
         [130, 164, 234],
         [116, 170, 236]],

        [[154, 184, 231],
         [136, 176, 239],
         [134, 174, 240],
         ...,
         [125, 159, 229],
         [125, 159, 229],
         [124, 176, 236]],

        [[148, 179, 226],
         [142, 182, 245],
         [137, 177, 243],
         ...,
         [135, 169, 239],
         [138, 172, 242],
         [126, 173, 226]]],
```

```
        [123, 138, 184]],

       [[251, 253, 254],
        [241, 246, 248],
        [224, 227, 233],
        ...,
        [106, 119, 163],
        [108, 120, 164],
        [106, 123, 166]],

       ...,

       [[ 49,  52,  57],
        [ 53,  56,  65],
        [ 60,  64,  79],
        ...,
        [104, 116, 173],
        [119, 134, 191],
        [143, 159, 204]],

       [[ 76,  78, 103],
        [ 89,  94, 116],
        [ 96, 100, 129],
        ...,
        [ 90,  98, 148],
        [100, 111, 161],
        [117, 126, 173]],

       [[113, 113, 149],
        [117, 118, 156],
        [116, 121, 160],
        ...,
        [ 70,  77, 116],
        [ 77,  84, 126],
        [ 86,  93, 142]]]], dtype=uint8)
```

In [25]:

```
x = x/255

x
```

Out[25]:

```
array([[[[0.58431373, 0.70196078, 0.85490196],
         [0.54117647, 0.71372549, 0.92156863],
         [0.55294118, 0.74509804, 0.98823529],
         ...,
         [0.6745098 , 0.77647059, 0.96078431],
         [0.70196078, 0.80784314, 0.99215686],
         [0.77254902, 0.81960784, 0.94509804]],

        [[0.59607843, 0.70980392, 0.8627451 ],
         [0.56078431, 0.73333333, 0.94117647],
         [0.50980392, 0.70196078, 0.94509804],
         ...,
         [0.69019608, 0.78431373, 0.94509804],
         [0.72156863, 0.80784314, 0.97254902],
         [0.78823529, 0.83921569, 0.96470588]],

        [[0.61960784, 0.7372549 , 0.89019608],
         [0.5372549 , 0.70980392, 0.91764706],
         [0.49803922, 0.69019608, 0.9372549 ],
         ...,
         [0.74509804, 0.81568627, 0.94901961],
         [0.76862745, 0.83137255, 0.96470588],
         [0.79607843, 0.84313725, 0.96862745]],

        ...,

        [[0.58431373, 0.70196078, 0.89019608],
         [0.51372549, 0.67058824, 0.91764706],
         [0.50588235, 0.6627451 , 0.91764706],
         ...,
         [0.50980392, 0.64313725, 0.91764706],
         [0.50980392, 0.64313725, 0.91764706],
         [0.45490196, 0.66666667, 0.9254902 ]],
```

```
      [[[1.        , 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        ...,
        [0.49803922, 0.5372549 , 0.72156863],
        [0.50588235, 0.54509804, 0.73333333],
        [0.52941176, 0.55686275, 0.74901961]],

       [[1.        , 0.99607843, 0.99215686],
        [0.99607843, 0.99607843, 0.99607843],
        [0.99607843, 0.99607843, 0.99607843],
        ...,
        [0.47058824, 0.50980392, 0.69411765],
        [0.4745098 , 0.51372549, 0.69803922],
        [0.48235294, 0.54117647, 0.72156863]],

       [[0.98431373, 0.99215686, 0.99607843],
        [0.94509804, 0.96470588, 0.97254902],
        [0.87843137, 0.89019608, 0.91372549],
        ...,
        [0.41568627, 0.46666667, 0.63921569],
        [0.42352941, 0.47058824, 0.64313725],
        [0.41568627, 0.48235294, 0.65098039]],

       ...,

       [[0.19215686, 0.20392157, 0.22352941],
        [0.20784314, 0.21960784, 0.25490196],
        [0.23529412, 0.25098039, 0.30980392],
        ...,
        [0.40784314, 0.45490196, 0.67843137],
        [0.46666667, 0.5254902 , 0.74901961],
        [0.56078431, 0.62352941, 0.8       ]],

       [[0.29803922, 0.30588235, 0.40392157],
        [0.34901961, 0.36862745, 0.45490196],
        [0.37647059, 0.39215686, 0.50588235],
        ...,
        [0.35294118, 0.38431373, 0.58039216],
        [0.39215686, 0.43529412, 0.63137255],
        [0.45882353, 0.49411765, 0.67843137]],

       [[0.44313725, 0.44313725, 0.58431373],
        [0.45882353, 0.4627451 , 0.61176471],
        [0.45490196, 0.4745098 , 0.62745098],
        ...,
        [0.2745098 , 0.30196078, 0.45490196],
        [0.30196078, 0.32941176, 0.49411765],
        [0.3372549 , 0.36470588, 0.55686275]]]])
```

In [26]:
```
y
```

Out[26]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0])
```

In [27]:
```
x.shape
```

Out[27]:
```
(131, 150, 150, 3)
```

In [28]:
```
model = Sequential()
model.add(Conv2D(64, (3,3), activation = 'relu' ))
model.add(MaxPooling2D((2,2)))
```

```
model.add(Conv2D(64,  (3,3),  activation = 'relu' ))
model.add(MaxPooling2D((2,2)))
```

In [29]:
```
model.add(Flatten())
```

In [30]:
```
model.add(Dense(128,  input_shape = x.shape[1:],  activation = 'relu'))
```

In [31]:
```
model.add(Dense(2,  activation = 'softmax'))
```

In [32]:
```
model.compile(optimizer = 'adam',  loss =
'sparse_categorical_crossentropy',
              metrics = ['accuracy'])
```

In [33]:
```
history = model.fit(x,y,epochs = 10,  validation_split = 0.2,batch_size
= 10,  callbacks =[tensorboard])  #batch_size = 32
```

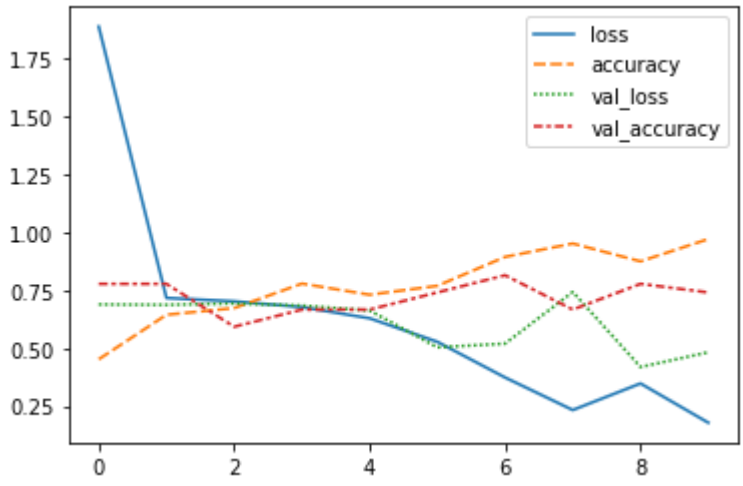
```
Epoch 1/10
 2/11 [====>.........................] - ETA: 10s - loss: 3.7444 - accuracy: 0.4000WA
RNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch
time (batch time: 0.6001s vs `on_train_batch_end` time: 1.6963s). Check your callback
s.
11/11 [==============================] - 8s 758ms/step - loss: 1.8891 - accuracy: 0.4
519 - val_loss: 0.6889 - val_accuracy: 0.7778
Epoch 2/10
11/11 [==============================] - 7s 600ms/step - loss: 0.7167 - accuracy: 0.6
442 - val_loss: 0.6872 - val_accuracy: 0.7778
Epoch 3/10
11/11 [==============================] - 6s 527ms/step - loss: 0.7023 - accuracy: 0.6
731 - val_loss: 0.6931 - val_accuracy: 0.5926
Epoch 4/10
11/11 [==============================] - 6s 554ms/step - loss: 0.6774 - accuracy: 0.7
788 - val_loss: 0.6838 - val_accuracy: 0.6667
Epoch 5/10
11/11 [==============================] - 6s 525ms/step - loss: 0.6290 - accuracy: 0.7
308 - val_loss: 0.6626 - val_accuracy: 0.6667
Epoch 6/10
11/11 [==============================] - 6s 550ms/step - loss: 0.5275 - accuracy: 0.7
692 - val_loss: 0.5035 - val_accuracy: 0.7407
Epoch 7/10
11/11 [==============================] - 6s 546ms/step - loss: 0.3734 - accuracy: 0.8
942 - val_loss: 0.5197 - val_accuracy: 0.8148
Epoch 8/10
11/11 [==============================] - 7s 623ms/step - loss: 0.2328 - accuracy: 0.9
519 - val_loss: 0.7425 - val_accuracy: 0.6667
Epoch 9/10
11/11 [==============================] - 6s 589ms/step - loss: 0.3476 - accuracy: 0.8
750 - val_loss: 0.4185 - val_accuracy: 0.7778
Epoch 10/10
11/11 [==============================] - 6s 546ms/step - loss: 0.1782 - accuracy: 0.9
712 - val_loss: 0.4823 - val_accuracy: 0.7407
```

In [34]:
```
history.history.keys()
```

Out[34]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [35]:
```
import seaborn as sns
sns.lineplot(data = history.history)
```

Out[35]:
```
<AxesSubplot:>
```


loss
accuracy
val_loss
val_accuracy
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
2
4
6
8

CHAPTER 8:
MODEL SUMMARY AND RESULT

- Four classification models were created- logistic regression, random forest, k-nearest neighbours, support vector machine for each of prediction of each diseases.the models were compared on the basis of prediction performance and the best model was chosen at the end.

- For the prediction of cervical cancer, the logistic regression model performs best among all the classification models created with the recall score of 0.94.

- For the prediction of Breast cancer, the Support Vector Machine model performs best among all the classification models created with the recall score of 0.93.

- For the prediction of Anemia, the Random Forest model performs best among all the classification models created with the recall score of 0.80.

- For the prediciton of Diabetes, The Support Vector machine model performs the best with recall score of 0.83.

- Although the dataset for the deep learning model of Oral cancer was small, the model performed with validation accuracy of 0.74

CHAPTER 9:
ADVANTAGES AND LIMITATIONS

ADVANTAGES -

- Due to increase amount of data growth in medical and healthcare field the accurate analysis on medical data which has been benefits from early patient care.

- With the help of disease data, data mining finds hidden pattern information in the huge amount of medical data.

- With the help of this machine learning models, better diagnosis strategies can be found out.

LIMITATIONS -

- The collected data may have wrong data entries which can severely affect the performance of model. Missing values are imputed with different strategies however this makes the model to learn from biases and not the actual data.

- Building the model on smaller dataset cannot predict the new values correctly thus reducing the performance of the model.

- Two many features in the dataset can overfit the machine learning models. This shows high accuracy while training the dataset but lacks the performance while testing and with new values.

CHAPTER 10:
FUTURE SCOPE

- This model can be made more accurate with addition of data overtime.
- Models can be created for prediction of other diseases in the same manner as the models created here.
- This models can be integrated into softwares with easy to understand user interface for the medical practitioners and doctors to get accurate diagnostic report.
- This models can further developed into a system which provides prescription drugs and other advices about diagnosis on the basis of data of medicines just like a general physician and a doctor.

CHAPTER 11:
CONCLUSION

- Predominant disease in the district of Gadchiroli were found out.
  It was found that Gadchiroli has Oral cancer cases are highest in India.
  Cervix and Breast are the major cancer sites in the women.

- Machine learning models performed with high accuracy. All the models predicting the disease with recall score of more than 0.90.

- This results of this projects directs towards the path of precise technology of machine learning and deep learning and usage of Artificial Intelligence in the field of Medicines and HealthCare.

Reference -

- Gadchiroli in Maharashtra records highest rate of oral cancer in India (https://www.hindustantimes.com/mumbai-news/gadchiroli-in-maharashtra-records-highest-rate-of-oral-cancer-in-india/story-5yhLdGJW9ldQhU3KW160sI.html)

- Oral cancer common in Gadchiroli: study (https://www.thehindu.com/news/national/other-states/oral-cancer-common-in-gadchiroli-study/article27006248.ece)

- Highest leprosy rate reported from Gadchiroli, Chandrapur districts (https://timesofindia.indiatimes.com/city/nagpur/highest-leprosy-rate- reported-from-gadchiroli-chandrapur-dists/articleshow/66961152.cms)

- From hypertension to cancers, alarm bells ringing in India's tribal belts (https://www.downtoearth.org.in/news/health/from-hypertension-to-cancers-alarm-bells-ringing-in-india-s-tribal-belts-61571)

- Prevalence of Anemia among Tribal Women of Reproductive Age (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4649888/)

- Cervical Cancer Dataset (https://archive.ics.uci.edu/ml/datasets/Cervical+cancer+%28Risk+Factors%29)

- Breast Cancer Dataset (https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28original%29)

- Oral Cancer Dataset (https://www.kaggle.com)