DIABETES DETECTOR USING MACHINE LEARNING ALGORITHMS

Machine Learning Project

SHRIVATS PODDAR 19BCE0750
KARTIK NAHTA 19BCE2009

Problem Statement

The Diabetes predicting system is a software application that will help predict whether or not a person is suffering from diabetes while assessing different health conditions such as; increased glucose level, age, family hereditary and blood pressure. Our project focuses on making machine learning models which can predict if a person suffers from diabetes while using specific health conditions as constraints. So different machine learning models will be used in order to get concluding results and whichever model serves to be most accurate will be used to improve the prediction system of diabetes amongst patients. The project uses several concepts of machine learning, and the data used are databases obtained from national authorities to test and verify the results. We will be taking these factors as constraints. These constraints are:

a) Pregnancies
b) Glucose
c) Blood Pressure
d) Skin Thickness
e) Insulin
f) BMI
g) Age
h) Diabetes pedigree function

By studying these constraints, we can predict the outcome as “Yes: Suffers from diabetes” or “No: Does not suffer from diabetes”. These data will be taken from an online database. We will be using different machine learning algorithms and compare each one so that we can provide the best outcome. Also, since pregnancy is also a constraint, the model works best for females.

Literature Review

[1] Kedar Pingale1, Sushant Surwase2, Kulkarni3, Saurabh Sarage4, Prof. Abhijeet Karve5: International Research Journal of Engineering and Technology (IRJET) 2019

The Disease Prediction system in [1] is based on predictive modelling that predicts the disease of the user in the basis of the symptoms that the user provides as an input to the system. The system analyses the symptoms provided by the user as input and gives the probability of the diseaseoutput. Disease Prediction is done by implementing the Naive Bayes Classifier. By using linear regression and decision trees we are predicting diseases like Diabetes, Malaria, Jaundice, Dengue, and Tuberculosis.This method uses machine learning algorithms for effective prediction of chronic disease outbreak in disease-frequent communities. We experiment the modified prediction models over real- life hospital data collected from central China. We propose a new convolutional neural net-work based multimodal disease risk prediction (CNN- MDRP) algorithm using structured and unstructured data from hospital.

[2] Ashish Kailash Pal, Pritam Rawal, Rahil Ruwala, Prof. Vaibhavi Patel, "Generic Disease Prediction using Symptoms with Supervised Machine Learning", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456- 3307, Volume 5 Issue 2, pp. 1082-1086, March-April 2019

Data Mining and Machine Learning in [2] plays the most inspiring areas of research that become most popular in health organizations. It also plays a vital part to uncover new patterns in the medicinal science and services association which thus accommodating for all the parties associated with this field. This project intends to form a diagnostic model of the common diseases based on the symptoms by using data mining techniques such as classification in the health domain. In this project, we are going to use algorithms like Random forest, Naive Bayes which can be utilized for health care diagnosis. Performances of the classifiers are compared to each other to find out the highest accuracy. This also helps us to find out persons who are affected by the infection. The test based on the outcomes of the diseases.

[3] International Journal of Engineering Research & Technology (IJERT) ISSN: 2278-0181 IJERTV4IS120622 www.ijert.org (This work is licensed under a Creative Commons Attribution 4.0 International License.) Vol. 4 Issue 12, December- 2015

Chronic kidney disease (CKD), also known as chronic renal disease. Chronic kidney disease [3] involves conditions that damage your kidneys and decrease their ability to keep you healthy. You may develop complications like high blood pressure, anemia (low blood count), weak bones, poor nutritional health, and nerve damage. . Early detection and treatment can often keep chronic kidney disease from getting worse. Data Mining is the term used for knowledge discovery from large databases. The task of data mining is to make use of historical data, to discover regular patterns and improve future decisions, follows from the convergence of several recent trends: the lessening cost of large data storage devices and the ever- increasing ease of collecting data over networks; the expansion of robust and efficient machine learning algorithms to process this data; and the lessening cost of computational power, enabling the use of computationally intensive methods for data analysis.

[4] Durai Vasan et al.; International Journal of Advance Research, Ideas and Innovations in Technology

Data Mining technologies have been widely used in the process of medical diagnosis and prognosis, extensively. These data mining techniques [4] have been used to analyze a colossal amount of medical data. The steep increase in the rate of obesity and an unhealthy lifestyle eventually reflects the likelihood and the frequent occurrence of liver-related diseases in the mass. In this project, the patient data sets are analyzed for the predictability of the subject to have a liver disease based purely on a widely analyzed classification model. Since there are pre-existing processes to analyze the patient data and the classifier data, the more important facet here is to predict the same conclusive result with a higher rate of accuracy. There are 5 distinct phases in this process. First, the min-max algorithm is applied to the original liver patient data set that could be collected from the UCI repository. In the second phase, significant attributes are demarcated by the use of PSO feature selection. This helps to bring out the subset of critical data, from the whole normalized datasets of liver patients.

[5] Ms. Priti V. Wadal, Dr. S. R. Gupta, Predictive Data Mining For Medical Diagnosis: An Overview Of Heart Disease Prediction, International Conference on Industrial Automation and Computing (ICIAC- 12- 13th April 2014).

A Systematic Review by Lauren N Carroll and Alan Au April 2014 Journal of Biomedical Informatics The objectives of this paper [5] were to: (1) identify public health user needs and preferences for infectious disease information visualization tools; (2) identify existing infectious disease information visualization tools and characterize their architecture and features; (3) identify commonalities among approaches applied to different data types; and (4) describe tool usability evaluation efforts and barriers to the adoption of such tools. The architecture of the tools was inconsistently described, and few tools in the review discussed the incorporation of usability studies or plans for dissemination.

[6] Pilot Study of a Novel Application for Data Visualization in Type 1 Diabetes by Jenise C. Wong, MD, PhD, Aaron B. Neinstein, MD, Howard Look, First Published February 8, 2017

A novel software application, Blip, was created to combine and display diabetes data from multiple devices in a uniform, user-friendly manner. The objective of this study [6] was to test the usability of this application by adults and caregivers of children with type 1 diabetes. Patients (n = 35) and caregivers of children with T1D (n = 30) using an insulin pump for >1 year ± CGM were given access to the software for 3 months. At baseline, 97% of participants agreed it was important for patients to know how to interpret glucose data. However, despite valuing shared responsibility, at baseline, 43% of participants never downloaded pump data, and only 9% did so at least once per month. At study end, 72% downloaded data at least once during the 3-month study, and 38% downloaded at least once per month. Regarding the software application, participants liked the central repository of data and the user interface. Suggestions included providing tools for understanding and interpreting glucose patterns, an easier uploading process, and access with mobile devices. Collaboration between developers and researchers prompted iterative, rapid development of data visualization software and improvements in the uploading process and user interface, which facilitates clinical integration and future clinical studies.

[7]Prediction Modeling Methodology by FrankJ.W.M.Dankers, AlbertoTraverso, LeonardWee, andSanderM.J.van Kuijk January 2019

A prediction model tries to stratify patients for their probability of having a certain outcome. The model [7] then allows you to identify patients that have an increased chance of an event and this may lead to treatment adaptations for the individual patient. The outcome variable of the prediction model can be anything, e.g., the risk of getting a side effect, the chance of surviving at a certain time point, or the probability of having a tumor recurrence. We can distinguish outcome variables into continuous variables or categorical variables. Continuous variables are described by numerical values and regression models are used to predict them. If the outcome has two categories this is referred to as binary classification and typical techniques are decision trees and logistic regression.

[8] A Model to Detect Heart Disease using Machine Learning Algorithm by O.E. Taylor, P. S. Ezekiel, F.B. Deedam-Okuchaba 2019, IJCSE International Journal of Computer Sciences and Engineering Open Access Vol.-7, Issue-11, Nov 2019 E-ISSN: 2347- 2693

Heart disease also refers to conditions that involve narrowed or blocked blood vessels that can lead to a heart attack, chest pain or stroke. This paper [8] presents a model for detecting heart disease using machine learning algorithm. The methodology adopted in this research is Agile Methodology, which follows planning, requirements analysis, designing, coding, testing and documentation in parallel during the stage of the production process. In this paper a Heart Dataset was trained using four different machine learning algorithms (K-Nearest Neighbours Classifier, Support Vector Classifier, Decision Tree Classifier and Random Forest Classifier). This model was deployed to the web using flask (a python framework), it takes 13 inputs from the user in order to make prediction. The model is implemented using python programming language and flask. This paper uses a Decision Tree Classifier Algorithm and the results obtained from the prediction shows an accuracy of about 68.83%.

[9] Building predictive model by using python and r for enhancing consumer satisfaction and advantages Dr.V.V.Narendra Kumar, Dr.K.Kondaia , Regula Thirupathi Department of CSE/IT, 1& 2 St.Mary's Group of Institutions, Highercollege of Tech,Muscat IJATES MARCH 2017

In the deregulated markets [9] empowered consumers expect innovative and personalized services. Hence there is a great need for the enterprises to develop innovative models to enhance the consumer satisfaction and thereby gain a competitive advantage over others. Applying Data mining and statistical techniques will aid us in developing several novel models. Predictive Analytics and predictive modelling can play a key role in optimizing consumer relation management. In this article we have enlightened the use of data mining techniques of business intelligence and the use data mining languages namely Python and R in developing predictive models.

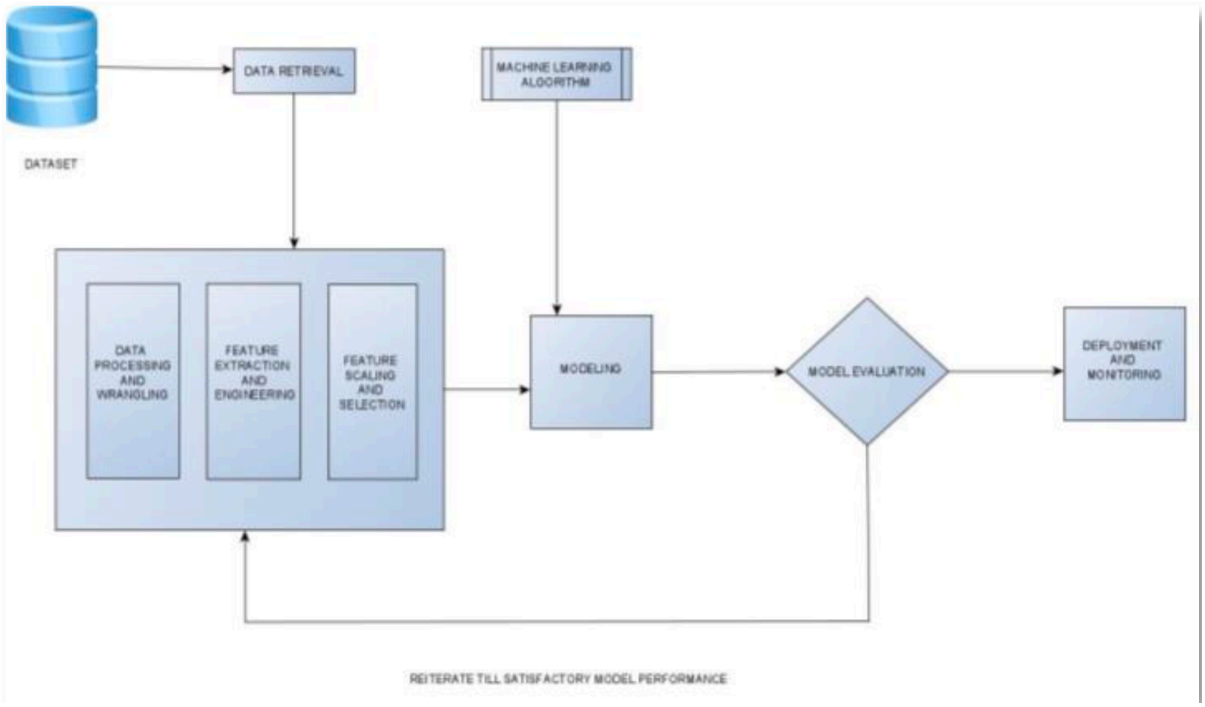
[10] Visualizing health practice to treat diabetes by Jeana H Frost and Brian K Smith January 2012 presented in Extended abstracts of the 2002 Conference on Human Factors in Computing Systems, CHI 2002, Minneapolis, Minnesota, USA

This research [10] is about how to help diabetics reflect upon and improve their own health practice by collecting and visualizing health related information. They introduced a new type of data collection to diabetics, photography, to complement the data they usually collect, blood sugar levels. Diabetics shoot pictures of meals, exercise, work, play and anything else they feel impacts health. We combine the quantitative glucose measurements with qualitative portraits of action into unified data visualizations. In doing so, we hope to make the relationship between physiology and behavior an object for discussion and reflection. More so, we hope that diabetics who viewed these data will begin to develop new interpretations of their lifestyles that will ultimately lead to healthier activities.

Methodology

Users enter information like, pregnancy, glucose, blood pressure, insulin, skin thickness, BMI, diabetes history and age, along with the option to choose the classification model. The algorithm then uses the training data that was given to it during initialization, and predicts possible graphs for the information entered by the patients and compares them with the data used in the initialization process. This allows the patient to compare their information with the standard data that is derived from online websites. Using this information, the likelihood of diabetes can be calculated and predictions can be made. Along with this an option to give feedback is available, in order to improve on the prediction method and on the overall

prediction. This all allows the prediction algorithm to learn and improve on the accuracy of the model.



- Data retrieval: It is the process of extracting data from the database for testing.
- Data Preprocessing and Wrangling: It is the process of cleaning and unifying messy and complex datasets for easy access and analysis, which includes mapping data from one raw format to another to organize data.
- Feature Extraction and Engineering: It is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing.
- Feature Scaling and Selection: It is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units.
- Modelling: It is the stage where the machine learning algorithm is applied and results are calculated.
- Model Evaluation: This phase of the project tests and prepares Machine Learning Models it for evaluation. Finally, at this stage, all the algorithms are compared and the optimum results corresponding to the best algorithm are chosen.
- Deployment and Monitoring: It is the stage where the final and optimum ML model is chosen and then accepted for use and monitoring.

Importing the Data set and Libraries Required:

To begin, we will import all of the libraries required for diabetes prediction. After that, we'll use the pandas library to import our data set. We shall now conduct exploratory data analysis

(EDA). It's a method of displaying, summarizing, and evaluating data that's concealed behind rows and columns.

Because the data contains numerous zeros and the values of glucose, blood pressure, skin thickness, insulin, and other attributes cannot be zero, the zeros are converted to mean and median values for the characteristics. The modified data set will then be exported. The Machine Learning Models are trained using the new Data set.

Data Visualization using the Different Plots:

To begin, a Pair plot is created. A Pair plot is a graph that depicts pairwise relationships in a dataset. The pair plot function creates a grid of Axes in which each variable in the data is shared across a single row and a single column on the y-axis. After that, a heat map of all the qualities is created. Gives the relationship between several qualities.

Following that, a Count plot of the entire data set aids in visualizing the number of people with diabetes against those who do not. To visualize the changes, we plot the Histogram and Box Plot for all of the attributes.

Using Different Machine Learning Algorithms:

Machine Learning Algorithms have been utilized in this paper which are mentioned as follows:

1. Decision Tree
2. K Nearest Neighbours Classifier
3. Logistic Regression
4. Random Forest
5. Support Vector Machine

Splitting, Training and Testing Dataset:

To begin, the data set is split into two sections: train (614[0.8]) and test (154[0.2]), with the model being trained and built on the training data set. We compare the outcome of the test data set with the model's result once the model has been developed. After that, we obtain the Model's Confusion Matrix and define the Model's Accuracy, Precision, and Recall on that basis.

On the basis of the results acquired, we will plot the Graphs for Machine Learning. The model's predictions for the various attributes fed into it for testing are shown in these charts.

5.4 Validation and Comparison Of Different Machine Learning Algorithms.

In this module, we compare the accuracy, precision, and recall of all the models' algorithms. When we compared the results, we discovered that Logistic Regression is the best fit for our data. It has an accuracy of roughly 81 per cent, and with more training and the use of the K-fold Validation Method, we can improve our model's accuracy.

Code:

https://drive.google.com/drive/folders/1sDiASZxuqE15ISK1qH1uWqcqVmzh87OC?usp=sharing

Importing all the Required Libraries

In [20]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(color_codes =True)
import pickle
import joblib
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from  sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Importing the Dataset

In [5]:

```
data = pd.read_csv("data/diabetes.csv")
```

Preview of Dataset

In [4]:

```
data.head(20)
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.000000	35	125	33.6	0.627	50	1
1	1	85.0	66.000000	29	125	26.6	0.351	31	0
2	8	183.0	64.000000	29	125	23.3	0.672	32	1
3	1	89.0	66.000000	23	94	28.1	0.167	21	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
4	0	137.0	40.000000	35	168	43.1	2.288	33	1
5	5	116.0	74.000000	29	125	25.6	0.201	30	0
6	3	78.0	50.000000	32	88	31.0	0.248	26	1
7	10	115.0	72.405184	29	125	35.3	0.134	29	0
8	2	197.0	70.000000	45	543	30.5	0.158	53	1
9	8	125.0	96.000000	29	125	32.3	0.232	54	1
10	4	110.0	92.000000	29	125	37.6	0.191	30	0
11	10	168.0	74.000000	29	125	38.0	0.537	34	1
12	10	139.0	80.000000	29	125	27.1	1.441	57	0
13	1	189.0	60.000000	23	846	30.1	0.398	59	1
14	5	166.0	72.000000	19	175	25.8	0.587	51	1
15	7	100.0	72.405184	29	125	30.0	0.484	32	1
16	0	118.0	84.000000	47	230	45.8	0.551	31	1
17	7	107.0	74.000000	29	125	29.6	0.254	31	1

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
18	1	103.0	30.000000	38	83	43.3	0.183	33	0
19	1	115.0	70.000000	30	96	34.6	0.529	32	1

In [5]:

```
data.shape
```

Out[5]:

```
(768, 9)
```

In [6]:

```
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    float64
 2   BloodPressure             768 non-null    float64
 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(4), int64(5)
memory usage: 54.1 KB
```

EDA

In [7]:

```
data.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.000000	6.00000	17.00
Glucose	768.0	121.686763	30.435949	44.000	99.75000	117.000000	140.25000	199.00
BloodPressure	768.0	72.405184	12.096346	24.000	64.00000	72.202592	80.00000	122.00
SkinThickness	768.0	29.108073	8.791221	7.000	25.00000	29.000000	32.00000	99.00

	count	mean	std	min	25%	50%	75%	max
Insulin	768.0	140.671875	86.383060	14.000	121.50000	125.000000	127.25000	846.00
BMI	768.0	32.455208	6.875177	18.200	27.50000	32.300000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.372500	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.000000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.000000	1.00000	1.00

Removal of Zeros\ \ Since there are many zeros in data and values of 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI' cannot be zero, Therefore, Converiting Zeros into NaN value

In [8]:
```
data_zeros = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI']
```

In [9]:
```
data[data_zeros] = np.where((data[data_zeros] == 0), np.nan,
data[data_zeros])
```

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Handling Missing Values\ \ Filling NaN values with suitable mean and median values

In [11]:
```
data.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.000000	6.00000	17.00

	count	mean	std	min	25%	50%	75%	max
Glucose	768.0	121.676763	30.435949	44.000	99.75000	117.000000	140.250000	199.00
BloodPressure	768.0	72.405184	12.096346	24.000	64.00000	72.202592	80.00000	122.00
SkinThickness	768.0	29.108073	8.791221	7.000	25.00000	29.000000	32.00000	99.00
Insulin	768.0	140.671875	86.383060	14.000	121.50000	125.000000	127.250000	846.00
BMI	768.0	32.455208	6.875177	18.200	27.50000	32.300000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.372500	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.000000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.000000	1.00000	1.00

In [12]:

```
data['Glucose'] = data['Glucose'].fillna(data['Glucose'].mean())
# data.isnull().sum()

data['BloodPressure'] =
data['BloodPressure'].fillna(data['BloodPressure'].mean())
# data.isnull().sum()

print("Skin thickness ", data['SkinThickness'].mean(),
data['SkinThickness'].median())

data['SkinThickness'] =
data['SkinThickness'].fillna(data['SkinThickness'].median())
# data.isnull().sum()

print("insulin " ,data['Insulin'].mean(), data['Insulin'].median())

data['Insulin'] = data['Insulin'].fillna(data['Insulin'].median())
# data.isnull().sum()

print("bmi " ,data['BMI'].mean(), data['BMI'].median())

data['BMI'] = data['BMI'].fillna(data['BMI'].median())
# data.isnull().sum()
```

```
Skin thickness  29.108072916666668 29.0
insulin  140.671875 125.0
bmi  32.45520833333333 32.3
```

In [13]:

```
data.head(20)
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.000000	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.000000	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.000000	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.000000	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.000000	35.0	168.0	43.1	2.288	33	1
5	5	116.0	74.000000	29.0	125.0	25.6	0.201	30	0
6	3	78.0	50.000000	32.0	88.0	31.0	0.248	26	1
7	10	115.0	72.405184	29.0	125.0	35.3	0.134	29	0
8	2	197.0	70.000000	45.0	543.0	30.5	0.158	53	1
9	8	125.0	96.000000	29.0	125.0	32.3	0.232	54	1
10	4	110.0	92.000000	29.0	125.0	37.6	0.191	30	0
11	10	168.0	74.000000	29.0	125.0	38.0	0.537	34	1

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
12	10	139.0	80.000000	29.0	125.0	27.1	1.441	57	0
13	1	189.0	60.000000	23.0	846.0	30.1	0.398	59	1
14	5	166.0	72.000000	19.0	175.0	25.8	0.587	51	1
15	7	100.0	72.405184	29.0	125.0	30.0	0.484	32	1
16	0	118.0	84.000000	47.0	230.0	45.8	0.551	31	1
17	7	107.0	74.000000	29.0	125.0	29.6	0.254	31	1
18	1	103.0	30.000000	38.0	83.0	43.3	0.183	33	0
19	1	115.0	70.000000	30.0	96.0	34.6	0.529	32	1

NEW UPDATED DATASET

In [14]:

```
data.to_csv(r'updates_dataset.csv')
```

Pair Plot to see Distribution of all data at a time and dependencies

Heat Map\ \ Gives Relation of different attribute with each other

In [16]:

```
plt.figure(figsize=(12,10))
sns.heatmap(data.corr(), annot = True, cmap = "YlGnBu")
plt.show()
```

PLOTTING HISTOGRAM AND BOX PLOT FOR ALL ATRRIBUTES

In [ ]:

Machine Learning Models

1. **DECISION TREE**

Splitting Dataset into Train and Test.

Building and Training the Model.

In [6]:

```
X=data.iloc[:,0:8].values
y=data.iloc[:,8].values
#
from sklearn.preprocessing import LabelEncoder
y= LabelEncoder().fit_transform(y)
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
=train_test_split(X,y,test_size=0.2,random_state=0)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
#
from sklearn.preprocessing import StandardScaler
sc_X= StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.transform(X_test)

from sklearn.tree import DecisionTreeClassifier
classifier_tree = DecisionTreeClassifier(criterion
='entropy',random_state=0)
classifier_tree.fit(X_train,y_train)
#
pickle_out = open("classifier_tree.pkl", "wb")
pickle.dump(classifier_tree, pickle_out)
pickle_out.close()
(614, 8) (154, 8) (614,) (154,)
```

Training Completed

Confusion Matrix

In [7]:

```
Y_pred = classifier_tree.predict(X_test)

from  sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test,Y_pred)
print(cm1)
[[78 29]
 [18 29]]
```

ACCURACY, PRECISION, RECALL

In [35]:

```
print("Accuracy of the Model is :
",(cm1[0][0]+cm1[1][1])*100/(cm1[0][0]+cm1[1][1]+cm1[0][1]+cm1[1][0]))
print("Precision of the Model is : ",(cm1[0][0])*100/(cm1[0][0]+cm1[1][0]))
print("Recall of the Model is : ",(cm1[0][0])*100/(cm1[0][0]+cm1[0][1]))

acc_x1 =
(cm1[0][0]+cm1[1][1])*100/(cm1[0][0]+cm1[1][1]+cm1[0][1]+cm1[1][0])
pre_x1 = (cm1[0][0])*100/(cm1[0][0]+cm1[1][0])
rec_x1 = (cm1[0][0])*100/(cm1[0][0]+cm1[0][1])
```

```
Accuracy of the Model is :  69.48051948051948
Precision of the Model is :  81.25
Recall of the Model is :  72.89719626168224
```

PLOTTING DECISION TREE FOR THE DATASET

In [8]:

```
from sklearn.decomposition import PCA
pca= PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance= pca.explained_variance_ratio_
#
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion ='entropy',random_state=0)
classifier.fit(X_train,y_train)
#
Y_pred = classifier.predict(X_test)
#
from matplotlib.colors import ListedColormap
X_set,y_set=X_test,y_test
X1,X2= np.meshgrid(np.arange(start=X_set[:,0].min()-
1,stop=X_set[:,0].max()+1,step=0.01),np.arange(start=X_set[:,1].min()-
1,stop=X_set[:,1].max()+1,step=0.01))
plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).
reshape(X1.shape),alpha=0.75,cmap =ListedColormap(('red','green')))
plt.xlim(X1.min(),X1.max())
plt.ylim(X2.min(),X2.max())
for i,j in enumerate(np.unique(y_set)):

plt.scatter(X_set[y_set==j,0],X_set[y_set==j,1],c=ListedColormap(('red','gr
een'))(i),label=j)
plt.title('Decision Tree Model')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```

1. **K NEAREST NEIGHBORS CLASSIFIER**

Building and Training the Model.

In [9]:

```
X=data.iloc[:,0:8].values
y=data.iloc[:,8].values
#
from sklearn.preprocessing import LabelEncoder
y= LabelEncoder().fit_transform(y)
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
=train_test_split(X,y,test_size=0.2,random_state=0)
#
from sklearn.preprocessing import StandardScaler
sc_X= StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
classifier_knn = KNeighborsClassifier(n_neighbors=3,metric='minkowski',p=2)
classifier_knn.fit(X_train,y_train)

pickle_out = open("classifier_knn.pkl", "wb")
pickle.dump(classifier_knn, pickle_out)
pickle_out.close()
```

Training Completed

Confusion Matrix

In [10]:

```
Y_pred = classifier_knn.predict(X_test)
#
from  sklearn.metrics import confusion_matrix
cm2 = confusion_matrix(y_test,Y_pred)
print(cm2)
[[89 18]
 [16 31]]
```

ACCURACY, PRECISION, RECALL

In [11]:

```
print("Accuracy of the Model is :
",(cm2[0][0]+cm2[1][1])*100/(cm2[0][0]+cm2[1][1]+cm2[0][1]+cm2[1][0]))
print("Precision of the Model is : ",(cm2[0][0])*100/(cm2[0][0]+cm2[1][0]))
print("Recall of the Model is : ",(cm2[0][0])*100/(cm2[0][0]+cm2[0][1]))

acc_x2 =
(cm2[0][0]+cm2[1][1])*100/(cm2[0][0]+cm2[1][1]+cm2[0][1]+cm2[1][0])
pre_x2 = (cm2[0][0])*100/(cm2[0][0]+cm2[1][0])
rec_x2 = (cm2[0][0])*100/(cm2[0][0]+cm2[0][1])
Accuracy of the Model is :  77.92207792207792
Precision of the Model is :  84.76190476190476
Recall of the Model is :  83.17757009345794
```

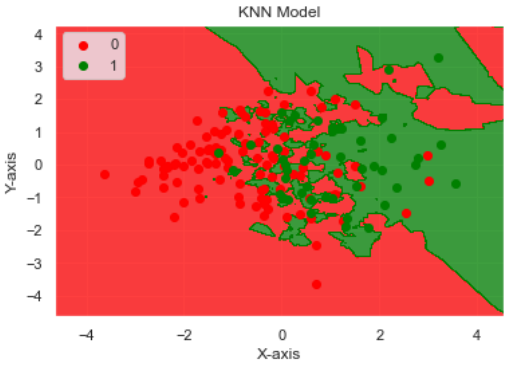
PLOTTING K-NEIGHBORS CLASSIFIER FOR THE DATASET

In [12]:

```
from sklearn.decomposition import PCA
pca= PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance= pca.explained_variance_ratio_
#
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3,metric='minkowski',p=2)
classifier.fit(X_train,y_train)
#
Y_pred = classifier.predict(X_test)
#
from matplotlib.colors import ListedColormap
X_set,y_set=X_test,y_test
X1,X2= np.meshgrid(np.arange(start=X_set[:,0].min()-
1,stop=X_set[:,0].max()+1,step=0.01),np.arange(start=X_set[:,1].min()-
1,stop=X_set[:,1].max()+1,step=0.01))
plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).
reshape(X1.shape),alpha=0.75,cmap =ListedColormap(('red','green')))
plt.xlim(X1.min(),X1.max())
plt.ylim(X2.min(),X2.max())
for i,j in enumerate(np.unique(y_set)):

plt.scatter(X_set[y_set==j,0],X_set[y_set==j,1],c=ListedColormap(('red','gr
een'))(i),label=j)
plt.title('KNN Model')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.show()
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
```

```
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```



1. **LOGISTIC REGRESSION**

Building and Training the Model.

In [14]:

```
data.head(20)
```

Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.000000	35	125	33.6	0.627	50	1
1	1	85.0	66.000000	29	125	26.6	0.351	31	0
2	8	183.0	64.000000	29	125	23.3	0.672	32	1
3	1	89.0	66.000000	23	94	28.1	0.167	21	0
4	0	137.0	40.000000	35	168	43.1	2.288	33	1
5	5	116.0	74.000000	29	125	25.6	0.201	30	0
6	3	78.0	50.000000	32	88	31.0	0.248	26	1

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
7	10	115.0	72.405184	29	125	35.3	0.134	29	0
8	2	197.0	70.000000	45	543	30.5	0.158	53	1
9	8	125.0	96.000000	29	125	32.3	0.232	54	1
10	4	110.0	92.000000	29	125	37.6	0.191	30	0
11	10	168.0	74.000000	29	125	38.0	0.537	34	1
12	10	139.0	80.000000	29	125	27.1	1.441	57	0
13	1	189.0	60.000000	23	846	30.1	0.398	59	1
14	5	166.0	72.000000	19	175	25.8	0.587	51	1
15	7	100.0	72.405184	29	125	30.0	0.484	32	1
16	0	118.0	84.000000	47	230	45.8	0.551	31	1
17	7	107.0	74.000000	29	125	29.6	0.254	31	1
18	1	103.0	30.000000	38	83	43.3	0.183	33	0
19	1	115.0	70.000000	30	96	34.6	0.529	32	1

In [15]:

```
data.describe()
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	29.108073	140.671875	32.455208	0.471876	33.240885	0.348958
std	3.369578	30.435949	12.096346	8.791221	86.383060	6.875177	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	25.000000	121.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.202592	29.000000	125.000000	32.300000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [16]:

```
X=data.iloc[:,0:8].values
y=data.iloc[:,8].values
#
from sklearn.preprocessing import LabelEncoder
y= LabelEncoder().fit_transform(y)
#print(X)
#print(y)
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
=train_test_split(X,y,test_size=0.2,random_state=0)
#
'''
from sklearn.preprocessing import StandardScaler
sc_X= StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.transform(X_test)
print("after",X_test)
```

```
'''
from sklearn.linear_model import LogisticRegression
classifier_log = LogisticRegression()
classifier_log.fit(X_train,y_train)
#
pickle_out = open("classifier_log.pkl", "wb")
pickle.dump(classifier_log, pickle_out)
pickle_out.close()
/usr/local/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:81
4: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

Confusion Matrix

In [17]:

```
Y_pred = classifier_log.predict(X_test)
print(Y_pred)
from sklearn.metrics import confusion_matrix
cm3 = confusion_matrix(y_test,Y_pred)
print(cm3)
[1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0
 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0
 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 1 0 0]
[[95 12]
 [20 27]]
```

ACCURACY, PRECISION, RECALL

In [18]:

```
print("Accuracy of the Model is :
",(cm3[0][0]+cm3[1][1])*100/(cm3[0][0]+cm3[1][1]+cm3[0][1]+cm3[1][0]))
print("Precision of the Model is : ",(cm3[0][0])*100/(cm3[0][0]+cm3[1][0]))
print("Recall of the Model is : ",(cm3[0][0])*100/(cm3[0][0]+cm3[0][1]))

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, Y_pred))

acc_x3 =
(cm3[0][0]+cm3[1][1])*100/(cm3[0][0]+cm3[1][1]+cm3[0][1]+cm3[1][0])
pre_x3 = (cm3[0][0])*100/(cm3[0][0]+cm3[1][0])
rec_x3 = (cm3[0][0])*100/(cm3[0][0]+cm3[0][1])
Accuracy of the Model is :  79.22077922077922
Precision of the Model is :  82.6086956521739
Recall of the Model is :  88.78504672897196
0.7922077922077922
```

PLOTTING LOGISTIC REGRESSION FOR THE DATASET

In [39]:

```
from sklearn.decomposition import PCA
pca= PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance= pca.explained_variance_ratio_
#
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train,y_train)
#
Y_pred = classifier.predict(X_test)
#
from matplotlib.colors import ListedColormap
X_set,y_set=X_test,y_test
X1,X2= np.meshgrid(np.arange(start=X_set[:,0].min()-
1,stop=X_set[:,0].max()+1,step=0.01),np.arange(start=X_set[:,1].min()-
1,stop=X_set[:,1].max()+1,step=0.01))
plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).
reshape(X1.shape),alpha=0.75,cmap =ListedColormap(('red','green')))
plt.xlim(X1.min(),X1.max())
plt.ylim(X2.min(),X2.max())
for i,j in enumerate(np.unique(y_set)):

plt.scatter(X_set[y_set==j,0],X_set[y_set==j,1],c=ListedColormap(('red','gr
een'))(i),label=j)
plt.title('Logistic Regression Model')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```

In [38]:

```
import joblib

def load_model(model_file):
        loaded_model = joblib.load("classifier_log.pkl","rb")
        return loaded_model


feature_list = [-0.54480808, -0.48575468,  0.11279888,  0.08828139, -
0.47396847,  0.13914985, -0.1876381,  -0.88240283]
single_sample = np.array(feature_list).reshape(1,-1)
loaded_model = load_model("classifier_log.pkl")
prediction = loaded_model.predict(single_sample)
pred_prob = loaded_model.predict_proba(single_sample)

print(prediction)
[0]
```

1. **RANDOM FOREST CLASSIFIER**

Building and Training the Model.

In [25]:

```
X=data.iloc[:,0:8].values
y=data.iloc[:,8].values
#
from sklearn.preprocessing import LabelEncoder
y= LabelEncoder().fit_transform(y)
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
=train_test_split(X,y,test_size=0.2,random_state=0)
#
from sklearn.preprocessing import StandardScaler
sc_X= StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier_random = RandomForestClassifier(n_estimators=10,
criterion='entropy',random_state=0)
classifier_random.fit(X_train,y_train)
#
pickle_out = open("classifier_random.pkl", "wb")
pickle.dump(classifier_random, pickle_out)
pickle_out.close()
```

Training Completed

Confusion Matrix

In [26]:

```
Y_pred = classifier_random.predict(X_test)
```

```
from  sklearn.metrics import confusion_matrix
cm5 = confusion_matrix(y_test,Y_pred)
print(cm5)
[[95 12]
 [22 25]]
```

ACCURACY, PRECISION, RECALL

In [27]:

```
print("Accuracy of the Model is :
",(cm5[0][0]+cm5[1][1])*100/(cm5[0][0]+cm5[1][1]+cm5[0][1]+cm5[1][0]))
print("Precision of the Model is : ",(cm5[0][0])*100/(cm5[0][0]+cm5[1][0]))
print("Recall of the Model is : ",(cm5[0][0])*100/(cm5[0][0]+cm5[0][1]))

acc_x5 =
(cm5[0][0]+cm5[1][1])*100/(cm5[0][0]+cm5[1][1]+cm5[0][1]+cm5[1][0])
pre_x5 = (cm5[0][0])*100/(cm5[0][0]+cm5[1][0])
rec_x5 = (cm5[0][0])*100/(cm5[0][0]+cm5[0][1])
Accuracy of the Model is :  77.92207792207792
Precision of the Model is :  81.19658119658119
Recall of the Model is :  88.78504672897196
```

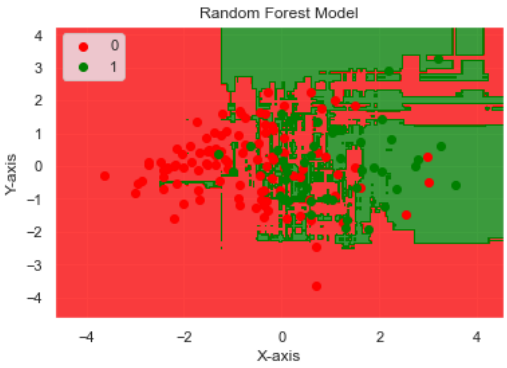
PLOTTING RANDOM FOREST CLASSIFIER FOR THE DATASET

In [28]:

```
from sklearn.decomposition import PCA
pca= PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance= pca.explained_variance_ratio_
#
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=10,
criterion='entropy',random_state=0)
classifier.fit(X_train,y_train)
#
Y_pred = classifier.predict(X_test)
#
from matplotlib.colors import ListedColormap
X_set,y_set=X_test,y_test
X1,X2= np.meshgrid(np.arange(start=X_set[:,0].min()-
1,stop=X_set[:,0].max()+1,step=0.01),np.arange(start=X_set[:,1].min()-
1,stop=X_set[:,1].max()+1,step=0.01))
plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).
reshape(X1.shape),alpha=0.75,cmap =ListedColormap(('red','green')))
plt.xlim(X1.min(),X1.max())
plt.ylim(X2.min(),X2.max())
for i,j in enumerate(np.unique(y_set)):

plt.scatter(X_set[y_set==j,0],X_set[y_set==j,1],c=ListedColormap(('red','gr
een'))(i),label=j)
plt.title('Random Forest Model')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
```

```
plt.savefig("rishabh.png")
plt.show()
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```



1. **SVM LINEAR**

Building and Training the Model.

In [29]:

```
X=data.iloc[:,0:8].values
y=data.iloc[:,8].values
#
from sklearn.preprocessing import LabelEncoder
y= LabelEncoder().fit_transform(y)
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
=train_test_split(X,y,test_size=0.2,random_state=0)
#
from sklearn.preprocessing import StandardScaler
sc_X= StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.transform(X_test)

from sklearn.svm import SVC
classifier_svm = SVC(kernel ='linear', random_state=0, probability = True)
classifier_svm.fit(X_train,y_train)
#
pickle_out = open("classifier_svm.pkl", "wb")
pickle.dump(classifier_svm, pickle_out)
pickle_out.close()
```

Training Completed

Confusion Matrix

In [30]:

```
Y_pred = classifier_svm.predict(X_test)
#
from  sklearn.metrics import confusion_matrix
cm6 = confusion_matrix(y_test,Y_pred)
print(cm6)
[[97 10]
 [20 27]]
```

ACCURACY, PRECISION, RECALL

In [31]:

```
print("Accuracy of the Model is :
",(cm6[0][0]+cm6[1][1])*100/(cm6[0][0]+cm6[1][1]+cm6[0][1]+cm6[1][0]))
print("Precision of the Model is : ",(cm6[0][0])*100/(cm6[0][0]+cm6[1][0]))
print("Recall of the Model is : ",(cm6[0][0])*100/(cm6[0][0]+cm6[0][1]))
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, Y_pred))

acc_x6 =
(cm6[0][0]+cm6[1][1])*100/(cm6[0][0]+cm6[1][1]+cm6[0][1]+cm6[1][0])
pre_x6 = (cm6[0][0])*100/(cm6[0][0]+cm6[1][0])
rec_x6 = (cm6[0][0])*100/(cm6[0][0]+cm6[0][1])
Accuracy of the Model is :  80.51948051948052
Precision of the Model is :  82.90598290598291
Recall of the Model is :  90.65420560747664
0.8051948051948052
```

PLOTTING SVM LINEAR FOR THE DATASET

In [32]:

```
from sklearn.decomposition import PCA
pca= PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance= pca.explained_variance_ratio_
#
from sklearn.svm import SVC
classifier = SVC(kernel ='linear', random_state=0)
classifier.fit(X_train,y_train)
#
Y_pred = classifier.predict(X_test)
#
from matplotlib.colors import ListedColormap
X_set,y_set=X_test,y_test
X1,X2= np.meshgrid(np.arange(start=X_set[:,0].min()-
1,stop=X_set[:,0].max()+1,step=0.01),np.arange(start=X_set[:,1].min()-
1,stop=X_set[:,1].max()+1,step=0.01))
plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).
reshape(X1.shape),alpha=0.75,cmap =ListedColormap(('red','green')))
plt.xlim(X1.min(),X1.max())
plt.ylim(X2.min(),X2.max())
```

```
for i,j in enumerate(np.unique(y_set)):

plt.scatter(X_set[y_set==j,0],X_set[y_set==j,1],c=ListedColormap(('red','gr
een'))(i),label=j)
plt.title('SVM Linear Model')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.show()
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```

COMAPRISION TABLE FOR ALL THE MACHINE LEARNING ALGORITHM

In [55]:

```
acc = [acc_x1, acc_x2, acc_x3, acc_x5, acc_x6]
pre = [pre_x1, pre_x2, pre_x3, pre_x5, pre_x6]
rec = [rec_x1, rec_x2, rec_x3, rec_x5, rec_x6]
ml = ["Decision Tree", "KNN", "Logistic Regression", "Random Forest",
"SVM"]

print("Accuracy:")
for i in range(5):
      print(acc[i], ":\t", ml[i])
print()
print("Precision:")
for i in range(5):
      print(pre[i], ":\t", ml[i])
print()
print("Recall:")
for i in range(5):
      print(rec[i], ":\t", ml[i])
Accuracy:
69.48051948051948 :     Decision Tree
77.92207792207792 :     KNN
79.22077922077922 :     Logistic Regression
77.92207792207792 :     Random Forest
80.51948051948052 :     SVM

Precision:
81.25 :  Decision Tree
84.76190476190476 :     KNN
82.6086956521739 :      Logistic Regression
81.19658119658119 :     Random Forest
```

```
82.90598290598291 :     SVM

Recall:
72.89719626168224 :     Decision Tree
83.17757009345794 :     KNN
88.78504672897196 :     Logistic Regression
88.78504672897196 :     Random Forest
90.65420560747664 :     SVM
```