Fergusson College (Autonomous)

Department of Statistics

T.Y. B.Sc. Project 2020-21


**Project By:**

1. Srishti Batra-4507
2. Akhilesh Deshmukh-4510
3. Akshay Deshmukh-4511
4. Sumit Gawande-4521
5. Amisha Matkar-4540
6. Chandrahas Nhayade-4546
7. Pratiksha Patil-4551
8. Snehal Patil-4553

# ACKNOWLEDGEMENT

We would like to express special thanks of gratitude to our guide, Mr. Shital Gadekar Sir, for his valuable inputs and continued motivation. Sir lent us valuable books, referred us to some great papers, gave new perspectives and ideas and was always accessible for discussion and doubts. We are profoundly grateful for the support.

This project would have been impossible without Josh Stammer from “Stat Quest” and Krish Naik on You tube. Their tutorials on Machine Learning techniques helped us to clear our basics, enabling us to take up this project. Also, a big thanks to Stack Exchange and Cross Validated online communities.

We are thankful to the Statistics Department, Fergusson College, for giving us this opportunity. This project truly helped us to apply statistics in real life, we have learnt much more than we ever imagine.

At starting, in this situation it was difficult for us to communicate with each other and discuss things due to online mode of communication, but it became easier and interesting by doing this project.

# Index

# INTRODUCTION

Diabetes is a disease which stays with the patient life-long except in some cases where the diabetes is gestational which occurs during pregnancy and often goes back to normal after the delivery. Typically, there are three types of diabetes. The management of diabetes mostly depend on patient himself or herself because in diabetes it is all about self-care. But of course, the guidance comes from the nurse and doctors and they need to educate the patients in order to control their condition. In order to prevent (in cases where there are chances of diabetes occurrence) or control (in cases where the person already have diabetes) it is very essential to take care of patient's weight, blood pressure, blood sugar and blood lipids.

**Types of diabetes: -**

There are three main types of diabetes:

**Type-I** diabetes used to be called juvenile-onset diabetes or insulin dependent diabetes. It is usually caused by an auto-immune reaction where the body's defence system attacks the cells that produce insulin. People with Type-I diabetes produce little or no insulin. The disease may affect people of any age, but usually develops in children or young adults. People with this form of diabetes need injections of insulin every day in order to control the levels of glucose in their blood. If people with Type-I diabetes do not have access to insulin, they will die.

**Type-II** diabetes used to be called non-insulin dependent diabetes or adult-onset diabetes, and accounts for at least 90% of all cases of diabetes. It is characterized by insulin resistance and relative insulin deficiency, either or both of which may be present at the time diabetes is diagnosed. The diagnosis of type-II diabetes can occur at any age. Type-II diabetes may remain undetected for many years and the diagnosis is often made when a complication appears, or a routine blood or urine glucose test is done. It is often, but not always, associated with overweight or obesity, which itself can cause insulin resistance and lead to high blood glucose levels. People with Type-II diabetes can often initially manage their condition through exercise and diet. However, over time most people will require oral drugs and or insulin.

**Gestational diabetes (GDM)** is a form of diabetes consisting of high blood glucose levels during pregnancy. It develops in one in 25 pregnancies worldwide and is associated with complications to both mother and baby. GDM usually disappears after pregnancy but women with GDM and their children are at an increased risk of developing Type-II diabetes later in life. Approximately half of women with a history of GDM go on to develop Type-II diabetes within five to ten years after delivery.

Table-1 shows the Normal Glucose Level chart.

**Table-1: Normal Glucose Level**

| **Blood sugar classification** | **Fasting Blood Sugar Levels** | **Post Meal Blood Sugar Level** |
|---|---|---|
| Normal | 70-100 mg/dl | 70-140 mg/dl |
| Pre-Diabetes | 101-125 mg/dl | 141-200 mg/dl |
| Diabetes | 125 mg/dl and above | 200 mg/dl and above |

# Machine Learning

In today's world we just want to make the computers more intelligent. Since the most basic requirement of intelligence is learning, hence came the subfield of AI that is called machine learning (ML). ML is one of the most rapidly evolving fields of AI, which is used in many areas of life, primarily in the healthcare field. ML has a great value in the healthcare field since it is an intelligent tool to analyse data, and the medical field is rich with data. In the past few years, numerous amounts of data were collected and stored because of the digital revolution. Monitoring and other data collection devices are available in modern hospitals and are being used every day, and abundant amount of data are being gathered. It is extremely hard or even impossible for humans to derive useful information from this massive amount of data. Hence machine learning is becoming increasingly popular to analyse such data and diagnose problems in the healthcare field. A simplified explanation of what the machine learning algorithms would do is, it will learn from previously diagnosed cases of patients. The resulting classifier can have many uses such as helping doctors to diagnosis new patients with higher speed and efficiency and training students and non-specialists to diagnose patients. Since we have vast number of medical datasets, machine learning can help us discover patterns and beneficial information from them. Although it has many uses, in the medical field, machine learning is mostly used for disease prediction. Many researchers became interested in using machine learning for diagnosing diseases because it helps to reduce diagnosing time and increases the accuracy and efficiency. Several diseases can be diagnosed using machine learning techniques, but the focus of this paper will be on diabetes diagnosis.

# Working of Models

Collecting the data

It involves the importation of data and useful libraries which we're going to use throughout the analysis.

Analysing the data

This step involves the analysis and exploration of data variables, in our case, skin thickness, pregnancy, etc.

Data wrangling

In this step, we remove the unnecessary contain such as null values in the data of respective variables. Basically, here we clear the data.

Train and test

Here we build the model and perform a split which divides the data in training and testing dataset.

Accuracy

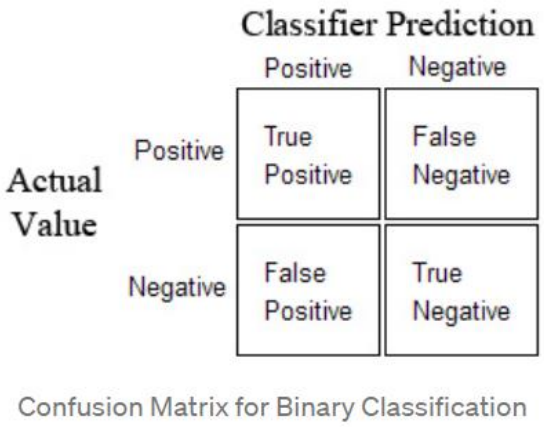
Here we test how much accurate our values are and find the accuracy.

## Some definitions:

Confusion Matrix

A confusion matrix is a table that often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

Confusion Matrix for (Scikit-learn documentation)




Confusion Matrix for Binary Classification

True positives (TP)

These are the correctly predicted positive values which means the value of actual class is Yes and the value of predicted class is also Yes.

True negatives (TN)

These are correctly predicted negative values which means that the value of actual class is No, and the value of predicted class is also No.

False Positives (FP)

False positive is an error in binary classification in which a test result incorrectly indicates the presence of a condition such as if the actual condition is false, but prediction tells it as True.

False Negatives (FN)

False negative is the error in binary classification in which a test result incorrectly fails to indicate the presence of the condition when it is present which means the actual class is Yes, but the predicted class is No.

Accuracy

Accuracy is the most intuitive performance measure, and it is simply a ratio of correctly predicted observations to total observations.

**Accuracy = (TP+TN)/(TP+FP+FN+TN)**

Specificity

It is the ability of a classifier to correctly classify the negative cases.

**Specificity = TN/(TN+FP) =>True Negative Rate (TNR)**

False Positive Rate (FPR)

It is defined as

**FPR = (FP)/(TN+FP)**

The false-positive rate is also known as false alarm and can be calculates as (1-Specificity).It can also be thought of as a plot of the power as a function of the Type-I Error of the decision rule.

False Negative Rate (FNR)

It measures the proportion of False Negatives i.e. test result incorrectly fails to indicate the presence of the condition when it is present which means the actual class is Yes, but the predicted class is No. It is like committing Type-II Error.

**FNR = (FN)/(TP+FN)**

Precision:

Precision tells us that how many of the correctly predicted cases turned out to be positive.

**Precision = (TP)/(TP+FP)**

Sensitivity (Recall)

It tells us that how many of the actual positive cases we were able to predict correctly with our model.

| **Sensitivity = TP/(TP+FN) =>True Positive Rate (TPR)** |
|---|

F1 Score

F1 score is the weighted average of precision and recall. Therefore, this score takes both false positives and false negatives into account.

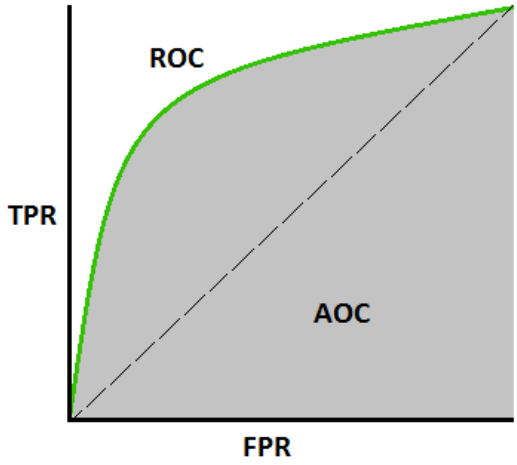
| **F1 Score = 2(Recall * Precision)/(Precision + Recall)=TP/((TP+(FP+FN)/2)** |
|---|

ROC-AUC Curve

In Machine Learning, performance measurement is an essential task. So, when it comes to a classification problem, we can count on an AUC - ROC Curve. When we need to check or visualize the performance of the multi-class classification problem, we use the AUC (**Area Under the Curve**) ROC (**Receiver Operating Characteristics**) curve. It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (**Area Under the Receiver Operating Characteristics**).

**What is the AUC - ROC Curve?**

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represent 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. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease. The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.




**How to speculate about the performance of How the model?**

An excellent model has AUC near to the 1 which means it has a good measure of separability. A poor model has AUC near to the 0 which means it has the worst measure of separability. In fact, it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means the model has no class separation capacity whatsoever.

# MOTIVATION

Diabetes is a chronic condition in which the body develops a resistance to insulin a hormone which balances your blood glucose levels. Approximately 247 million people all over the world have been diagnosed with diabetes. Increased caution is required for lean people with type 2 diabetes at every age, as type 1 diabetes can occur in older people and with slower onset than is normally seen in the young.

As per International Diabetes Federation (IDF), India is one of the participants with more diabetic people. It is reported that about 72 million people are affected by diabetes in 2017-18. Though the diabetes is not fatal, it is not curable, and the ill-effects continue till the lifetime of patient. Hence, it is better to predict rather than treat the disease. There are three types of diabetes. When patient is affected by diabetes of type 1, then the insulin is not secreted by the body such that patient must inject insulin to the body. In case of type 2 diabetes, the body develops resistivity against insulin and insulin is not exploited in usual way. The type 3 diabetes occurs due to excessive blood glucose levels.

Though the diabetes is not fatal, it is not curable, and the ill-effects continue till the lifetime of patient. Hence, it is better to predict rather than treat the disease. We wanted to use emerging techniques to model this data, and in the process demystify all the buzzwords that have cropped up in recent times- data science, machine learning, analytics, artificial intelligence as well as more accurate ones such as Random forests.

# ABSTRACT

In this project we attempt to predict diabetes in patients by using different statistical models such as Naive Bayes, K-Nearest Neighbours, Random Forest, Logistic regression, Decision Trees, Support Vector Machine(linear and radial). There are 9 variables used in this classification which are pregnancies (no of times pregnant), glucose, blood pressure, skin thickness, BMI, insulin, age, diabetes pedigree function, and outcome. The 7 models are compared using metrics such as accuracy, AUC, sensitivity and specificity. In the process we obtain their confusion matrices and ROC curves. Our aim is to find the model which is best suited for predicting this data. Exploratory analysis has also been done to get a feel for the data and to detect any obvious relationships that might exist.

# KEYWORDS

Machine learning, Confusion matrix, Sensitivity, Specificity, Accuracy, ROC, AUC, Naive Bayes, K-Nearest Neighbours, Random Forest, Logistic regression, Decision Trees, Support Vector Machine.

# OBJECTIVES

The main objective of our project is to build a machine learning model to accurately predict whether the patients in the dataset have diabetes or not.

We will build and train following machine learning models during this analysis.

1. Naive Bayes
2. K-Nearest Neighbours
3. Random Forest
4. Logistic Regression
5. Decision Trees
6. Support Vector Machine

# DATA SET

The data set is comprised of 768 observations and 9 variables such as pregnancies, glucose, blood pressure, triceps, insulin, mass, pedigree, age and diabetes (outcome). We will use diabetes as our response or target variable. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to predict whether a patient has diabetes based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. All patients here are females at least 21 years old of Pima Indian heritage.

**Data description for the 9 variables is as follows,**

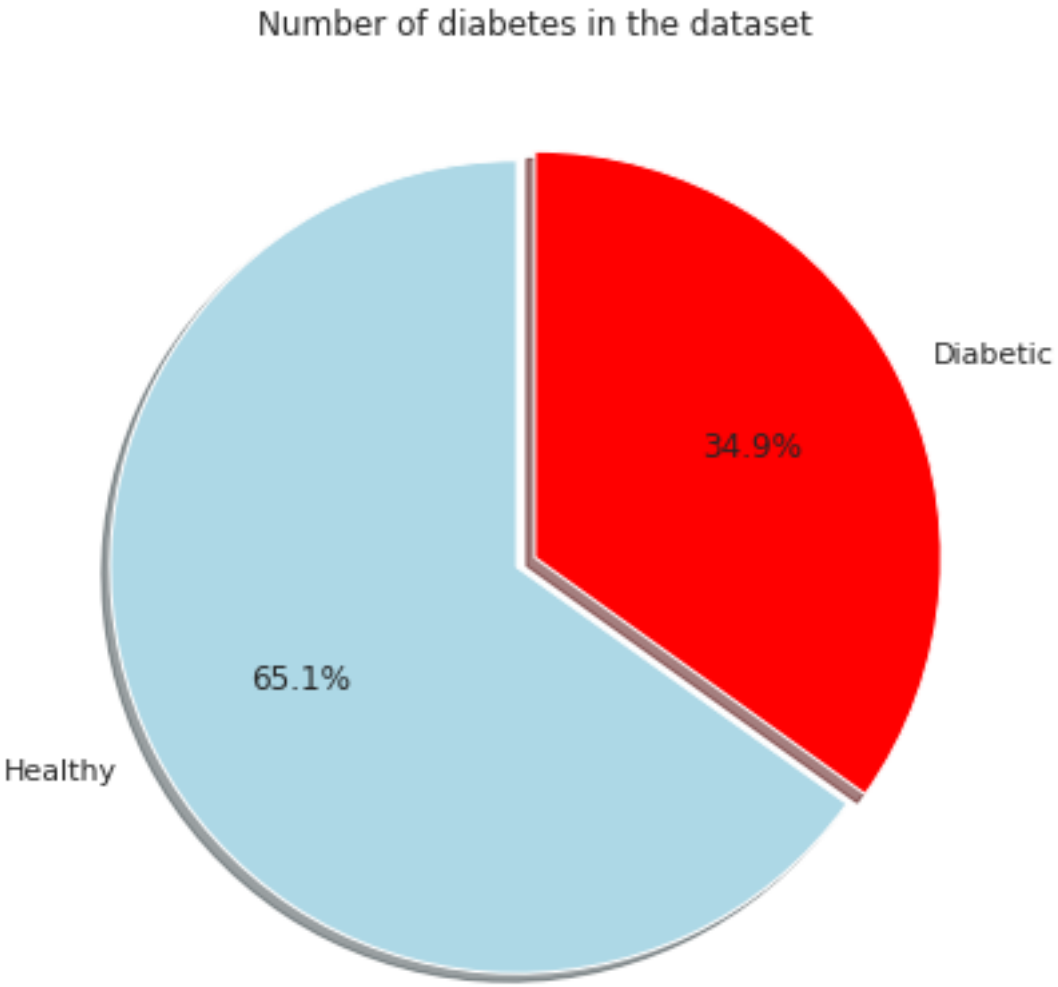
| Sr.No | ATTRIBUTES | MEANING | DATA TYPE |
|---|---|---|---|
| 1. | Pregnant | The total count of pregnancies encountered by the candidate. | Numeric |
| 2. | Glucose | The glucose concentration is measured by means of oral glucose tolerance test. | Numeric |
| 3. | Blood Pressure | The diastolic blood pressure encountered by the arteries during the pause time between the heart beats (mm Hg). | Numeric |
| 4. | Triceps | The thickness of skin measured by considering the subcutaneous fats (mm). | Numeric |
| 5. | Insulin | The insulin serum is considered for 2 hours (muU/ml). | Numeric |
| 6. | Mass | The BMI considering both height in (m^2) and weight in (kg) of patient. | Numeric |
| 7. | Diabetes Pedigree Function | This attribute considers the diabetes disease history with respect to the blood relations. | Numeric |
| 8. | Age | The age of patient. | Numeric |
| 9. | Diabetes | Class variables (test for diabetes) | Numeric |

More information on this dataset can be read by accessing the website:

http://math.furman.edu/~dcs/courses/math47/R/library/r

# EXPLORATORY DATA ANALYSIS

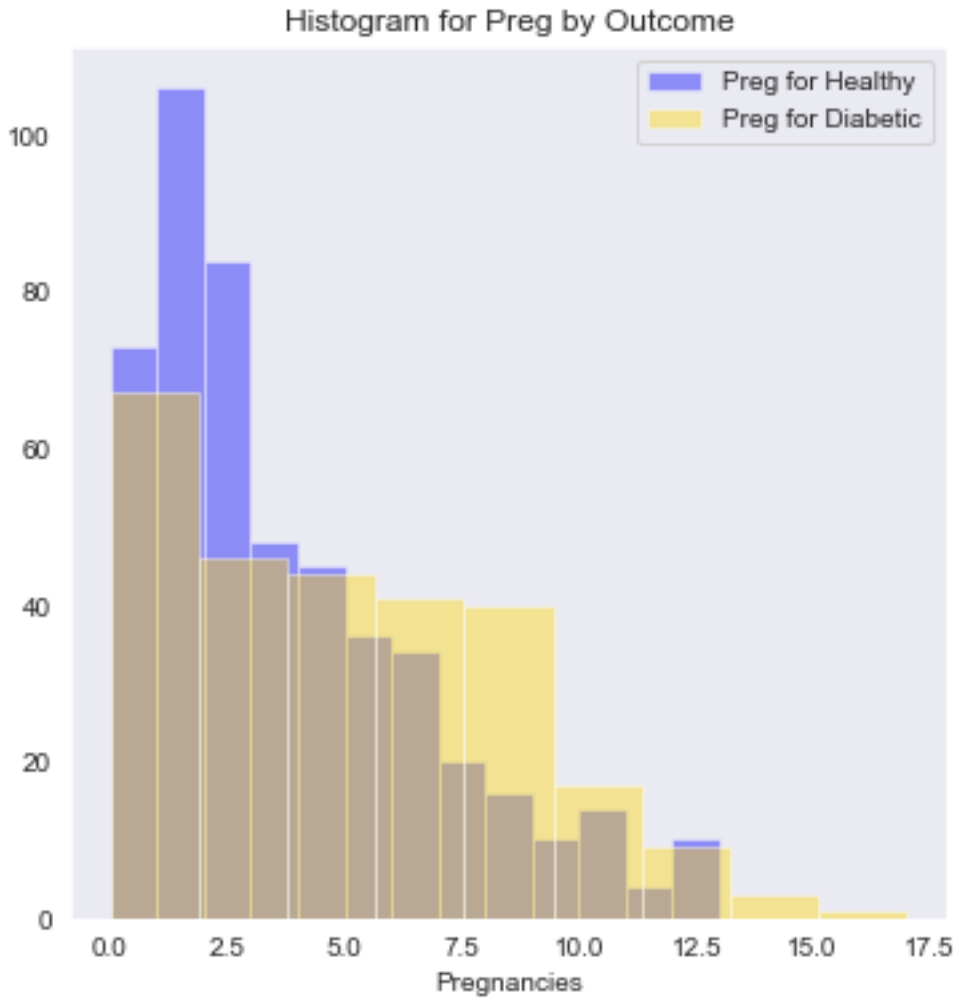
**PIE CHART**:




From above pie chart, we can say that around 65% of the people are Healthy and 35% are Diabetic.

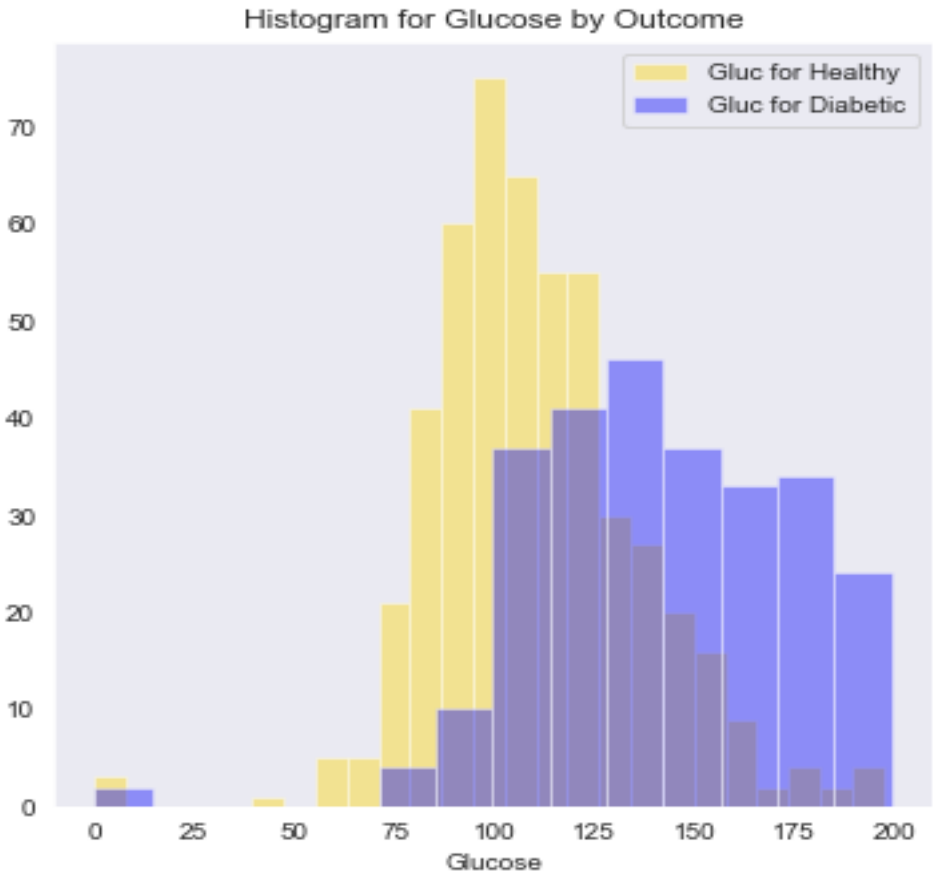
# DISTRIBUTION PLOTS

## Healthy vs. Diabetic by Pregnancy




From above graph, we can say that the as the pregnancies increase the number of diabetic people is more as compared to healthy people. Therefore higher number of pregnancies might be a contributing factor to diabetes.

# Healthy vs. Diabetic by Glucose

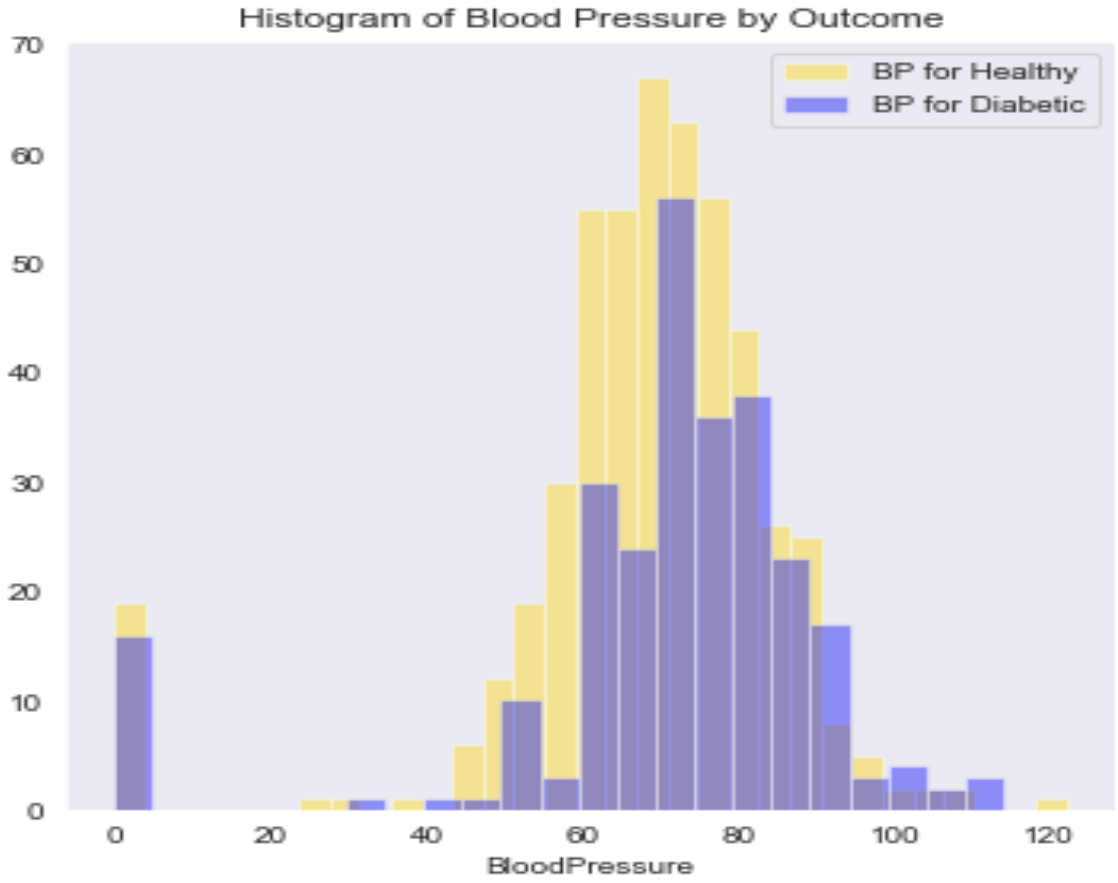



Diabetes is a disease that occurs when your blood glucose, also called blood sugar, is too high. Blood glucose is your main source of energy and comes from the food you eat.

The Glucose level for a Normal Adult is around 120-130mg/dl anything above it means that the person is likely suffering from pre-diabetes and diabetes.

From above graph, we can see the Healthy person are more around 120mg/dl but it then gradually drops, and for diabetic person it is vice versa. Therefore high glucose levels might be a cause for diabetes.

# Healthy vs. Diabetic by Blood Pressure

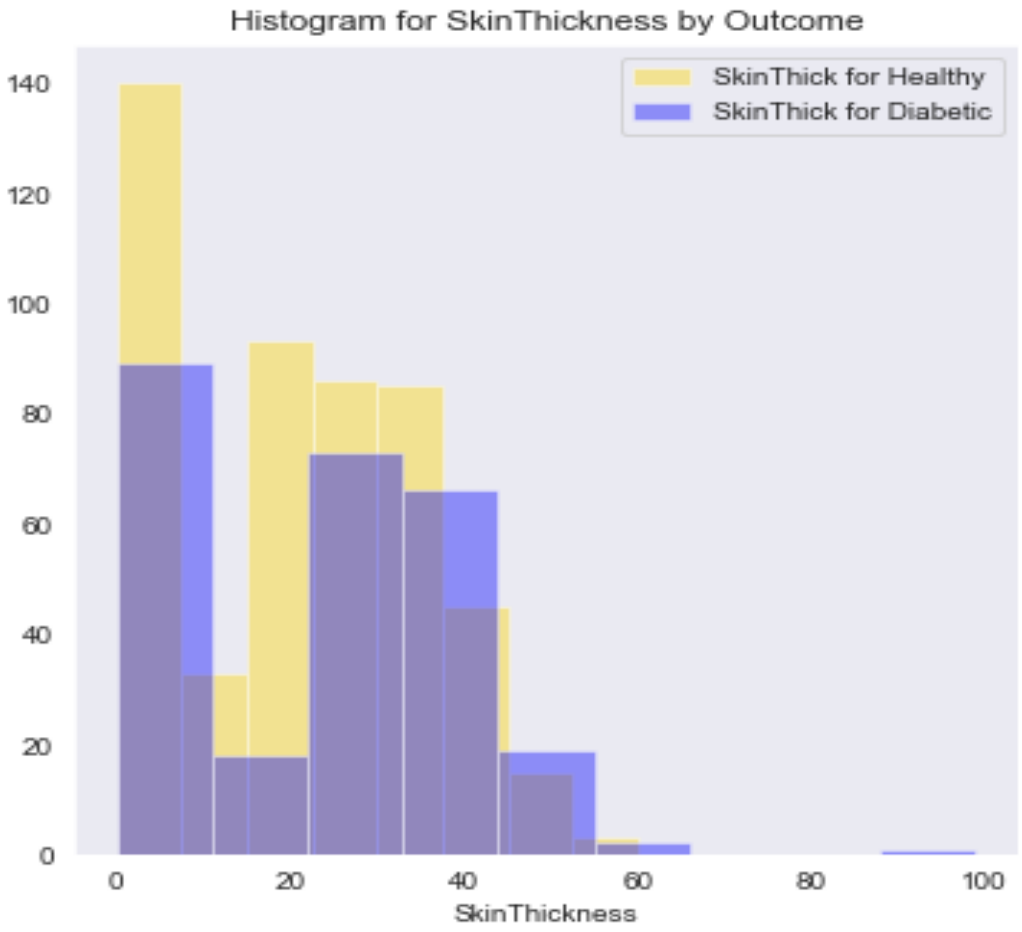



High blood pressure (also known as “hypertension”) is very common in people with diabetes. In fact, the two conditions often go hand-in-hand because they can both result from the same lifestyle factors.

Diabetes damages arteries and makes them targets for hardening, called atherosclerosis. That can cause high blood pressure, which if not treated, can lead to trouble including blood vessel damage, heart attack, and kidney failure. For a Normal person BP should be at or below 120/80 mm Hg, the person with hypertension can be above 139/89 mm Hg.

From above graph, we can say that diabetic and healthy people are evenly distributed with low and normal BP.

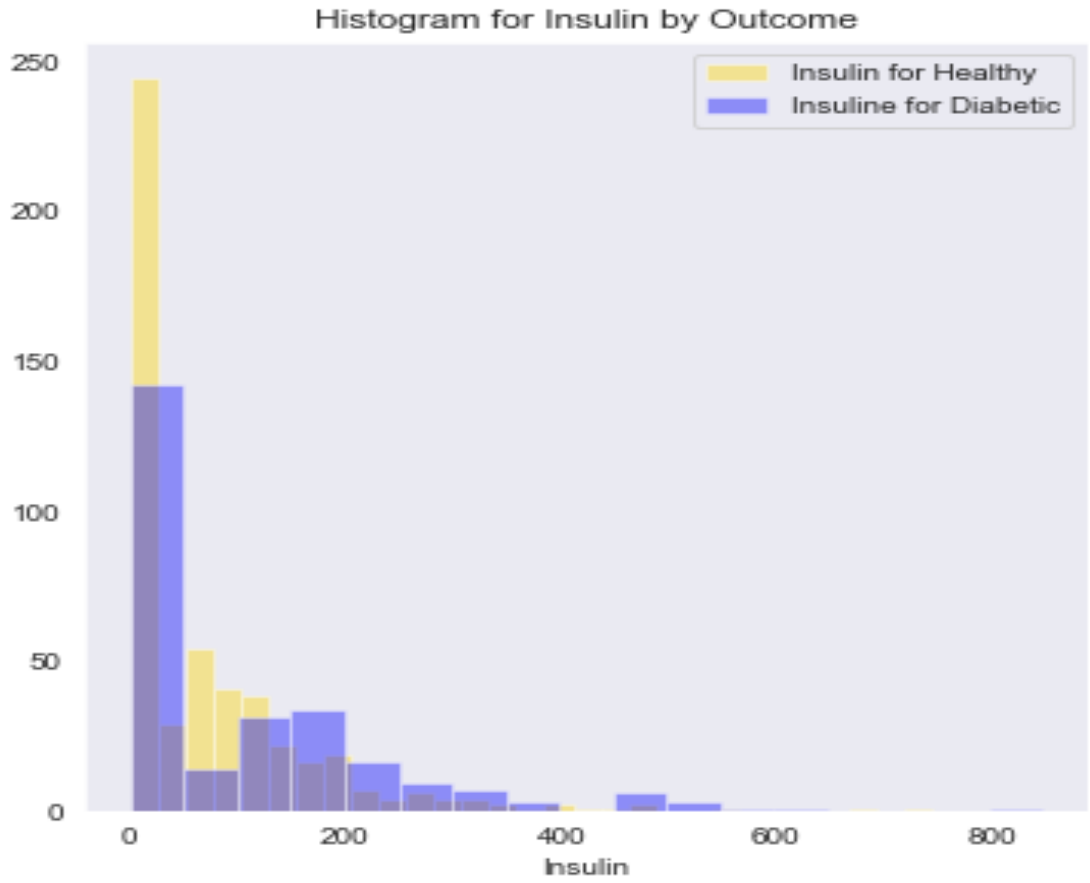
# Healthy vs. Diabetic by Skin Thickness




Changes to the blood vessels because of diabetes can cause a skin condition called diabetic dermopathy. Dermopathy appears as scaly patches that are light brown or red, often on the front of the legs. The patches do not hurt, blister, or itch, and treatment generally is not necessary.

From above graph, the distribution between healthy and diabetic people are around same for skin thickness.

# Healthy vs. Diabetic by Insulin

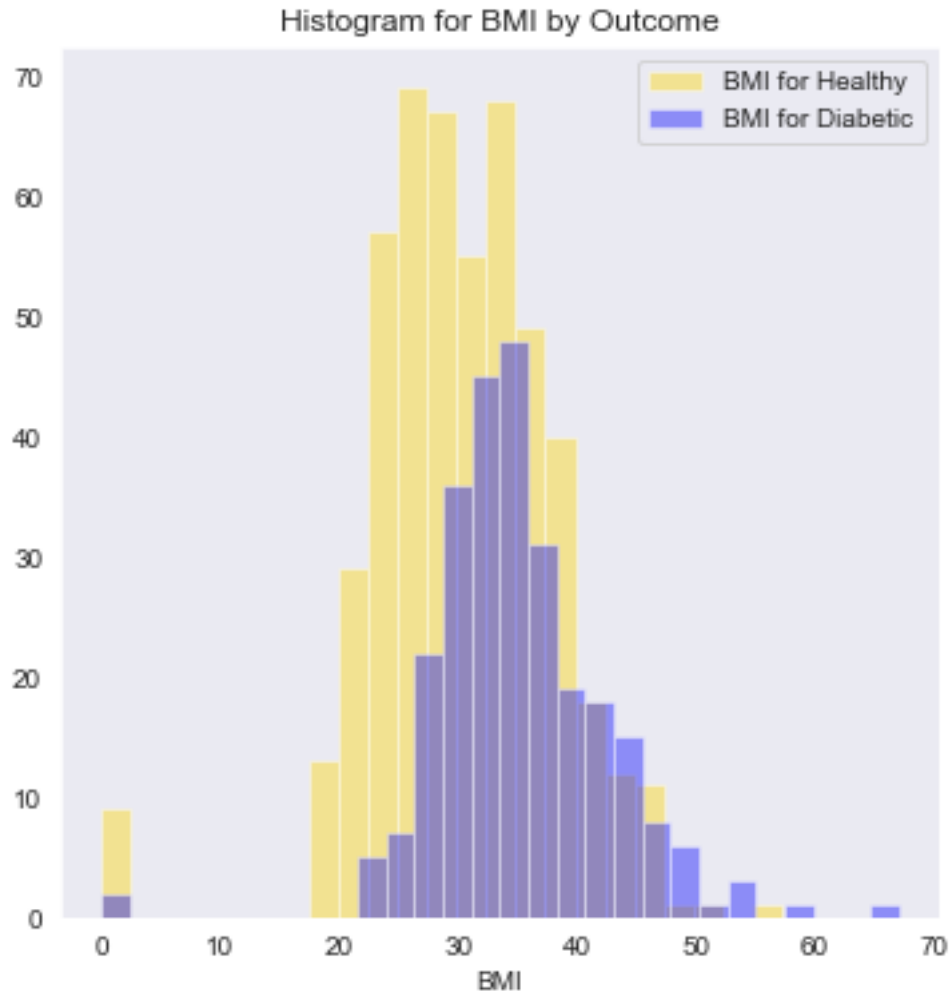



Insulin is a hormone that your pancreas makes to allow cells to use glucose. When your body isn't making or using insulin correctly, you can take man-made insulin to help control your blood sugar. Many types can be used to treat diabetes.

Insulin helps control blood glucose levels by signalling the liver and muscle and fat cells to take in glucose from the blood. Insulin therefore helps cells to take in glucose to be used for energy. If the body has sufficient energy, insulin signals the liver to take up glucose and store it as glycogen.

From above graph, we can see that as the insulin level increases the number of diabetic people is more as compared to the number of healthy people. High insulin level might be predisposing factor for diabetes. There are more healthy people around insulin levels 0-100.

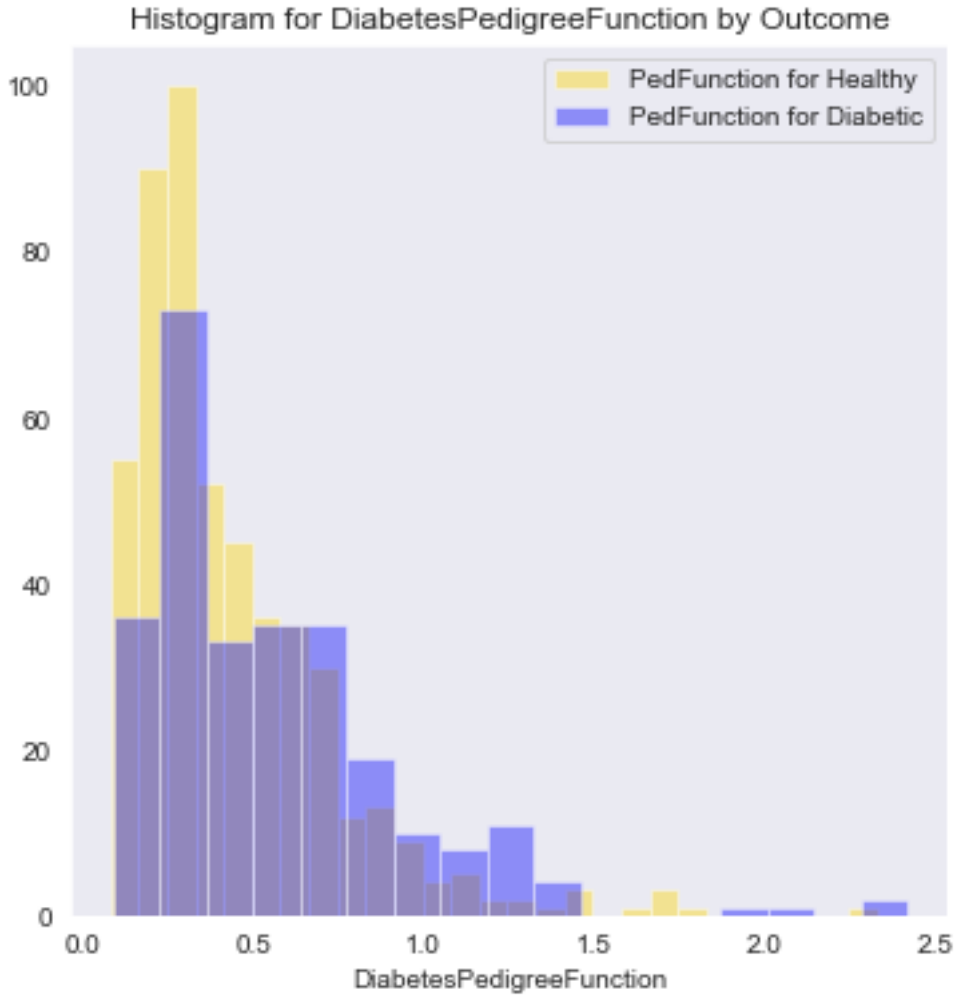
# Healthy vs. Diabetic by BMI




Being overweight (BMI of 25-29.9), or affected by obesity (BMI of 30-39.9) or morbid obesity (BMI of 40 or greater), greatly increases your risk of developing type 2 diabetes. The more excess weight you have, the more resistant your muscle and tissue cells become to your own insulin hormone.

From above graph we can say that, as the BMI increases the person likely being healthy decreases and being diabetic increases.

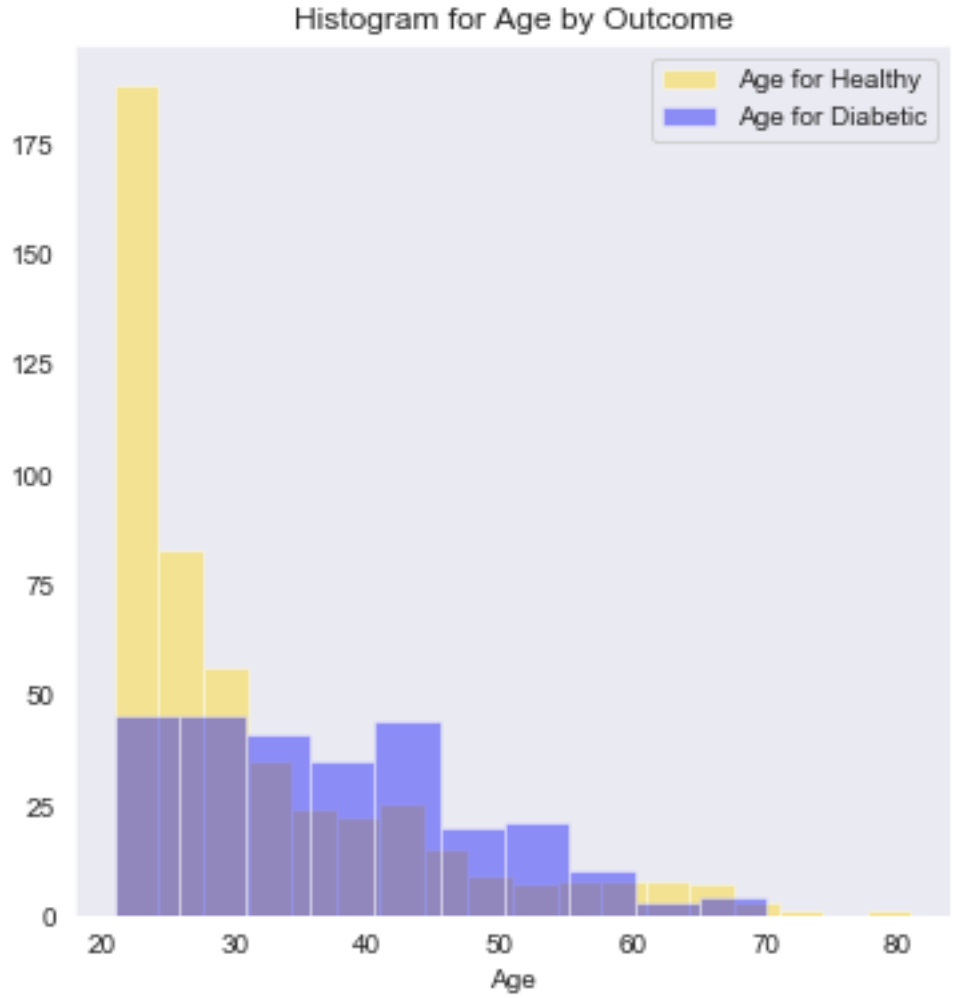
# Healthy vs. Diabetic by Diabetes Pedigree Function




Diabetes Pedigree Function is a function which scores likelihood of diabetes based on family history. It provided some data on diabetes mellitus history in relatives and the genetic relationship of those relatives to the patient.

From above graph, as the function increase the diabetic people increases, showing that the diabetes could be hereditary for an individual.

# Healthy vs. Diabetic by Age




As the person ages, they are at high risk for the development of type 2 diabetes due to the combined effects of increasing insulin resistance and impaired pancreatic islet function with aging.

From above graph, we can see that there are more healthy people around 20-25 age but as the age gradually increases so does the people being diabetic, this shows that age could be a contributing factor to diabetes.

# CLASSIFICATION MODELS

In [1]:
```
import warnings
warnings.filterwarnings('ignore')  # ignore warnings
import numpy as np                 #library used for working with arrays
import matplotlib.pyplot as plt    #create,plot,decorate figures
import seaborn as sns              #data visualization
import pandas as pd                #working with table data
%matplotlib inline
sns.set_style('darkgrid')          #setting styles
```

In [2]:
```
df=pd.read_csv('C:/Ak/1project-D/diabetes.csv')
df.head()
```

Out[2]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 |

In [3]:
```
print("Shape of Data is==>",df.shape)
```

```
Shape of Data is==> (768, 9)
```

In [4]:
```
df.info()          #information
```

```
<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
```

In [5]:
```
df.describe().T                    #description
```

Out[5]:

| | count | mean | std | min | 25% | 50% | 75% | |
|---|---|---|---|---|---|---|---|---|
| **Pregnancies** | 768.0 | 3.845052 | 3.369578 | 0.000 | 1.00000 | 3.0000 | 6.00000 | |
| **Glucose** | 768.0 | 120.894531 | 31.972618 | 0.000 | 99.00000 | 117.0000 | 140.25000 | 1 |
| **BloodPressure** | 768.0 | 69.105469 | 19.355807 | 0.000 | 62.00000 | 72.0000 | 80.00000 | 1 |
| **SkinThickness** | 768.0 | 20.536458 | 15.952218 | 0.000 | 0.00000 | 23.0000 | 32.00000 | |
| **Insulin** | 768.0 | 79.799479 | 115.244002 | 0.000 | 0.00000 | 30.5000 | 127.25000 | 8 |
| **BMI** | 768.0 | 31.992578 | 7.884160 | 0.000 | 27.30000 | 32.0000 | 36.60000 | |
| **DiabetesPedigreeFunction** | 768.0 | 0.471876 | 0.331329 | 0.078 | 0.24375 | 0.3725 | 0.62625 | |
| **Age** | 768.0 | 33.240885 | 11.760232 | 21.000 | 24.00000 | 29.0000 | 41.00000 | |
| **Outcome** | 768.0 | 0.348958 | 0.476951 | 0.000 | 0.00000 | 0.0000 | 1.00000 | |

In [6]:
```
for i in df.columns:                    #variable names
    print(i)
```

```
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
```

In [7]:
```
df.rename({'DiabetesPedigreeFunction':'DPF'},inplace=True,axis=1)
df.head()
```

Out[7]:

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

In [8]:
```
df.dtypes
```

Out[8]:
```
Pregnancies         int64
Glucose             int64
BloodPressure       int64
SkinThickness       int64
Insulin             int64
BMI               float64
DPF               float64
Age                 int64
Outcome             int64
dtype: object
```

#

📌Defining a function for Detecting and Exluding Outliers

#

In [9]:
```
def std_based(col_name,df):                    #taking range between mea
n- or +std
    mean = df[col_name].mean()
    std = df[col_name].std()
    cut_off = std * 3
    lower, upper = mean - cut_off, mean + cut_off
    new_df = df[(df[col_name] < upper) & (df[col_name] > lower)]
    return new_df
```

#

📌 Checking if there are any NULL values.

#

In [10]:
```
df.isnull().sum() #this will show the number of missing values.
```

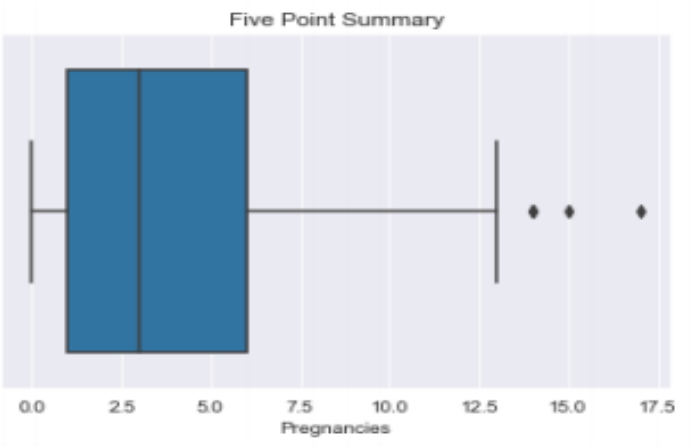
Out[10]:
```
Pregnancies       0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DPF               0
Age               0
Outcome           0
dtype: int64
```

In [11]:
```
df['Pregnancies'].describe()            #description
```

Out[11]:
```
count    768.000000
mean       3.845052
std        3.369578
min        0.000000
25%        1.000000
50%        3.000000
75%        6.000000
max       17.000000
Name: Pregnancies, dtype: float64
```

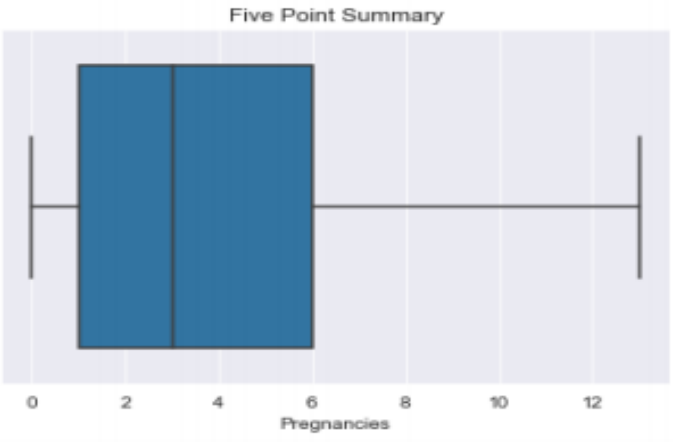
In [12]:
```
#sns.boxplot use to draw boxplots

sns.boxplot('Pregnancies',data=df).set_title('Five Point Summary')
```

Out[12]:
```
Text(0.5, 1.0, 'Five Point Summary')
```




In [13]:
```
#Treating outlier and verifying it

df = std_based('Pregnancies',df)

sns.boxplot('Pregnancies',data=df).set_title('Five Point Summary')
```

Out[13]:
```
Text(0.5, 1.0, 'Five Point Summary')
```

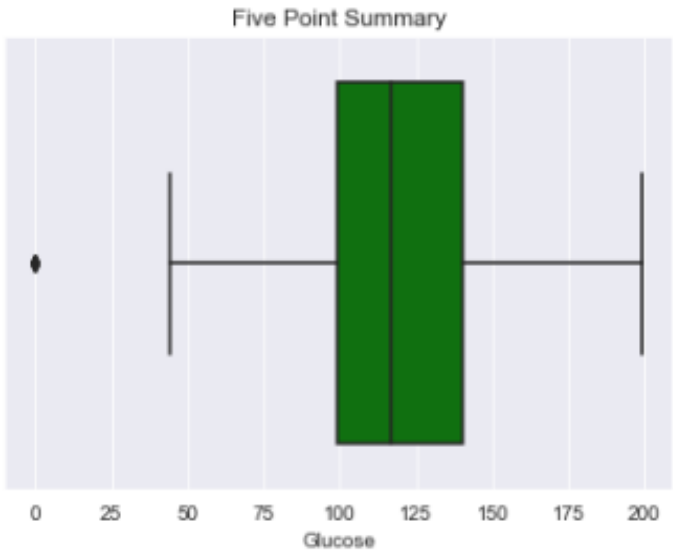


In [14]:
```
df['Glucose'].describe()
```

Out[14]:
```
count    764.000000
mean     120.776178
std       31.946234
min        0.000000
25%       99.000000
50%      117.000000
75%      140.000000
max      199.000000
Name: Glucose, dtype: float64
```

In [15]:
```
sns.boxplot('Glucose',data=df,color='green').set_title('Five Point Summary')
```

Out[15]:
```
Text(0.5, 1.0, 'Five Point Summary')
```



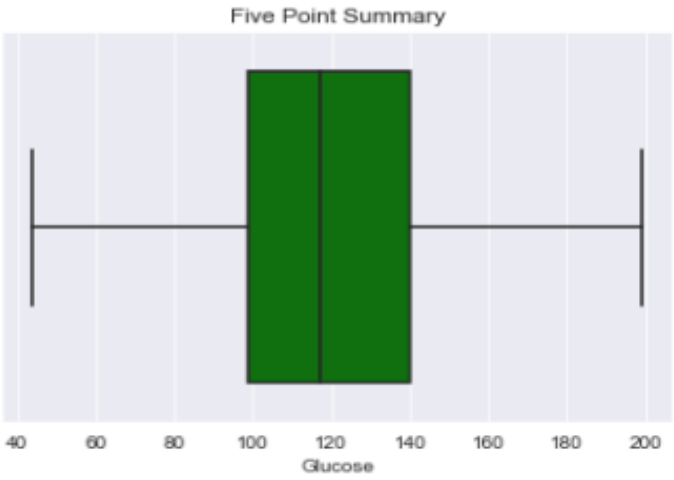

In [16]:
```
df.Glucose = df.Glucose.replace(0,df.Glucose.mean()) ##### 0 values are
 replaced by mean ######
df.head()
```

Out[16]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DPF | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85.0 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183.0 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89.0 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137.0 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

In [17]:
```
sns.boxplot('Glucose',data=df,color='g').set_title('Five Point Summary')
```

Out[17]:
```
Text(0.5, 1.0, 'Five Point Summary')
```

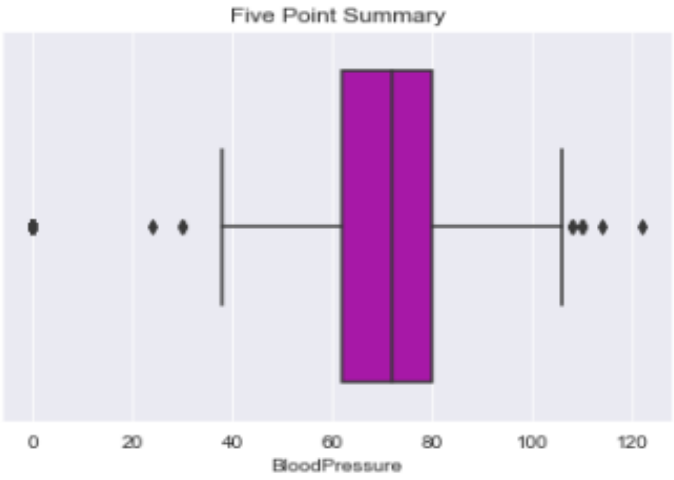



In [18]:
```
df.BloodPressure.describe()
```

Out[18]:
```
count    764.000000
mean      69.098168
std       19.401789
min        0.000000
25%       62.000000
50%       72.000000
75%       80.000000
max      122.000000
Name: BloodPressure, dtype: float64
```

In [19]:
```
sns.boxplot('BloodPressure',data=df,color='m').set_title('Five Point Sum
mary')
```

Out[19]:
```
Text(0.5, 1.0, 'Five Point Summary')
```



In [20]:
```
df.BloodPressure = df.BloodPressure.replace(0,df.BloodPressure.median())
df.head()
```

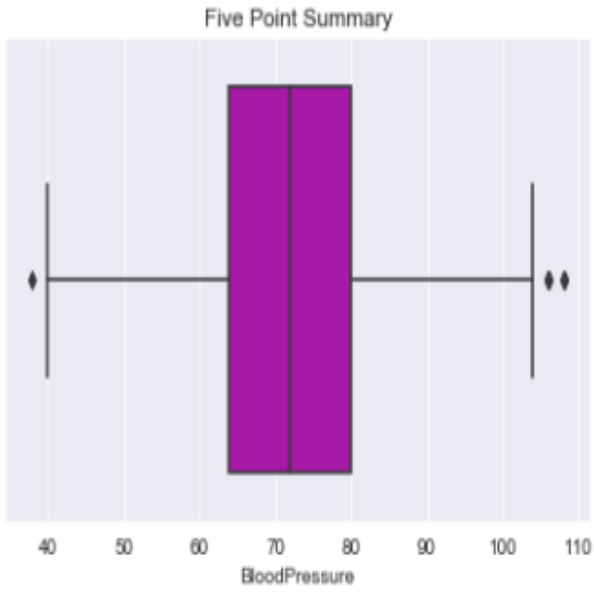
Out[20]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DPF | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85.0 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183.0 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89.0 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137.0 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

In [21]:
```
df  = std_based('BloodPressure',df)
```

In [22]:
```
sns.boxplot('BloodPressure',data=df,color='m').set_title('Five Point Summary')
```

Out[22]:
```
Text(0.5, 1.0, 'Five Point Summary')
```




In [23]:
```
df.SkinThickness.describe()
```

Out[23]:
```
count    756.000000
mean      20.428571
std       15.952377
min        0.000000
25%        0.000000
50%       23.000000
75%       32.000000
max       99.000000
Name: SkinThickness, dtype: float64
```

In [24]:
```
df.SkinThickness = df.SkinThickness.replace(0,df.SkinThickness.mean())
df.head()
```

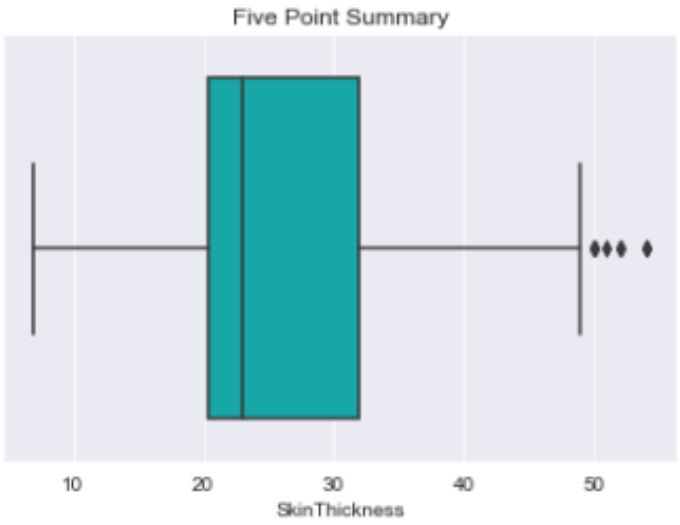
Out[24]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DPF | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72 | 35.000000 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85.0 | 66 | 29.000000 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183.0 | 64 | 20.428571 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89.0 | 66 | 23.000000 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137.0 | 40 | 35.000000 | 168 | 43.1 | 2.288 | 33 | 1 |

In [25]:
```
df = std_based("SkinThickness",df)
```

In [26]:
```
sns.boxplot('SkinThickness',data=df,color='c').set_title('Five Point Summary')
```

Out[26]:
```
Text(0.5, 1.0, 'Five Point Summary')
```

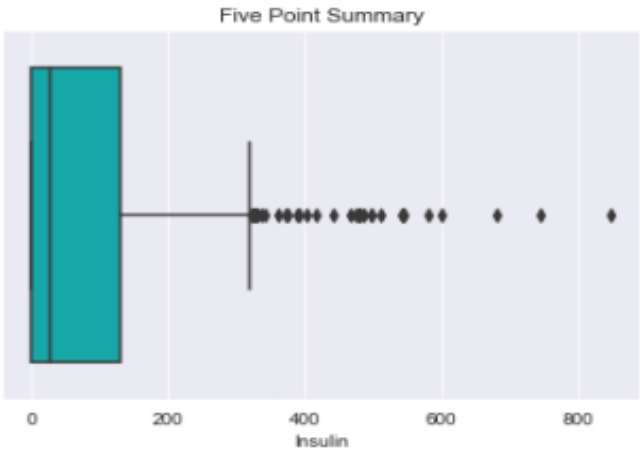



In [27]:
```
df.Insulin.describe()
```

Out[27]:
```
count    752.000000
mean      79.889628
std      115.995453
min        0.000000
25%        0.000000
50%       26.000000
75%      128.250000
max      846.000000
Name: Insulin, dtype: float64
```

In [28]:
```
sns.boxplot('Insulin',data=df,color='c').set_title('Five Point Summary')
```

Out[28]:
```
Text(0.5, 1.0, 'Five Point Summary')
```




In [29]:
```
df.Insulin = df.Insulin.replace(0,df.Insulin.median())
df.head()
```

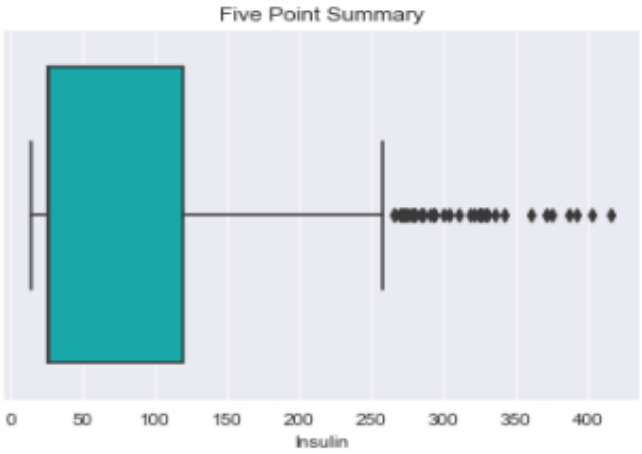
Out[29]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DPF | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72 | 35.000000 | 26 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85.0 | 66 | 29.000000 | 26 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183.0 | 64 | 20.428571 | 26 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89.0 | 66 | 23.000000 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137.0 | 40 | 35.000000 | 168 | 43.1 | 2.288 | 33 | 1 |

In [30]:
```
df = std_based('Insulin',df)
```

In [31]:
```
sns.boxplot('Insulin',data=df,color='c').set_title('Five Point Summary')
```

Out[31]:
```
Text(0.5, 1.0, 'Five Point Summary')
```

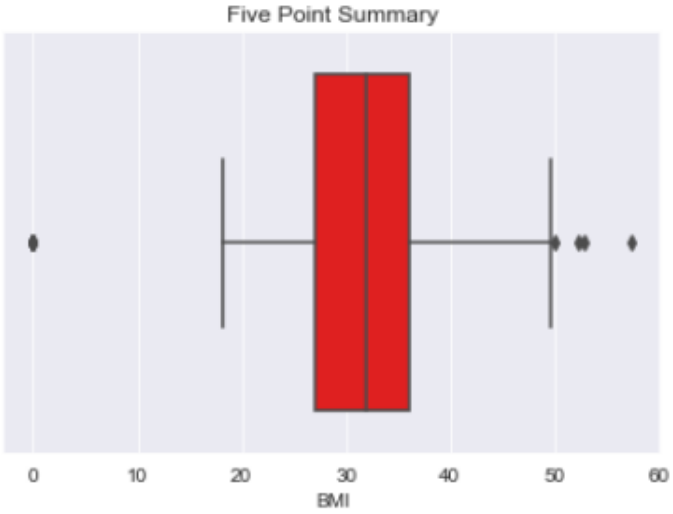


In [32]:
```
df.BMI.describe()
```

Out[32]:
```
count    734.000000
mean      31.649728
std        7.630830
min        0.000000
25%       27.025000
50%       32.000000
75%       36.100000
max       57.300000
Name: BMI, dtype: float64
```

In [33]:
```
sns.boxplot('BMI',data=df,color='r').set_title('Five Point Summary')
```

Out[33]:
```
Text(0.5, 1.0, 'Five Point Summary')
```



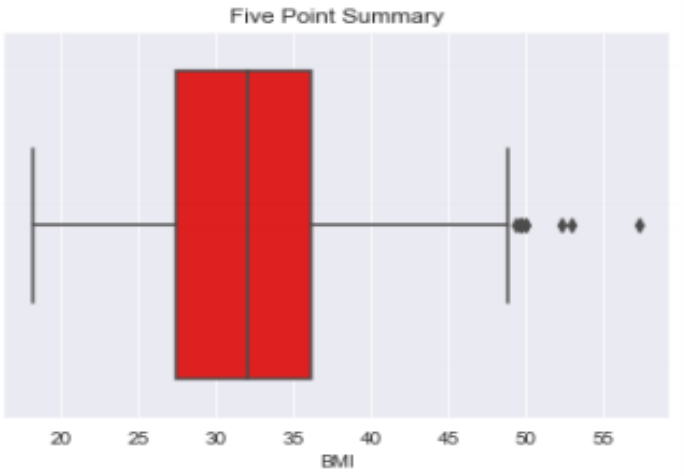

In [34]:
```
df.BMI = df.BMI.replace(0,df.BMI.mean())
df.head()
```

Out[34]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DPF | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72 | 35.000000 | 26 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85.0 | 66 | 29.000000 | 26 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183.0 | 64 | 20.428571 | 26 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89.0 | 66 | 23.000000 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137.0 | 40 | 35.000000 | 168 | 43.1 | 2.288 | 33 | 1 |

In [35]:
```
sns.boxplot('BMI',data=df,color='r').set_title('Five Point Summary')
```

Out[35]:
```
Text(0.5, 1.0, 'Five Point Summary')
```

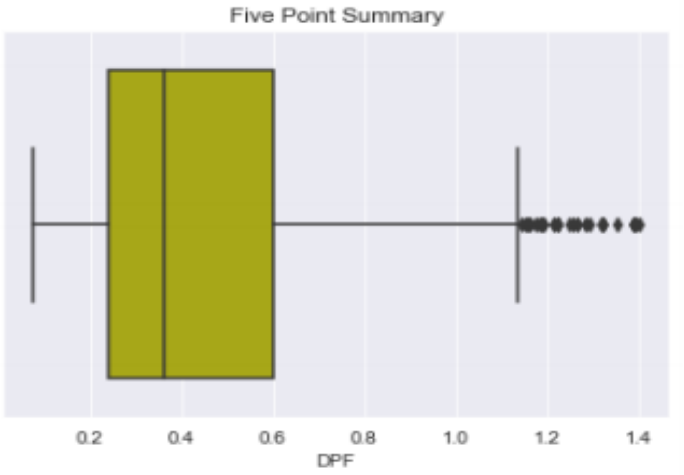



In [36]:
```
df.DPF.describe()
```

Out[36]:
```
count    734.000000
mean       0.464274
std        0.313676
min        0.078000
25%        0.244000
50%        0.367000
75%        0.612750
max        2.288000
Name: DPF, dtype: float64
```

In [37]:
```
df = std_based('DPF',df)

sns.boxplot('DPF',data=df,color='y').set_title('Five Point Summary')
```

Out[37]:
```
Text(0.5, 1.0, 'Five Point Summary')
```

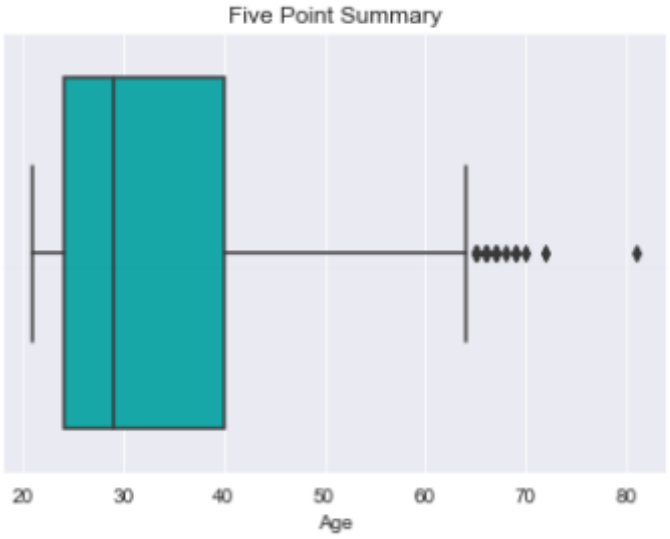


In [38]:
```
df.Age.describe()
```

Out[38]:
```
count    724.000000
mean      33.111878
std       11.711371
min       21.000000
25%       24.000000
50%       29.000000
75%       40.000000
max       81.000000
Name: Age, dtype: float64
```

In [39]:
```
sns.boxplot('Age',data=df,color='c').set_title('Five Point Summary')
```

Out[39]:
```
Text(0.5, 1.0, 'Five Point Summary')
```




In [40]:
```
df.head()
```

Out[40]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DPF | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72 | 35.000000 | 26 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85.0 | 66 | 29.000000 | 26 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183.0 | 64 | 20.428571 | 26 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89.0 | 66 | 23.000000 | 94 | 28.1 | 0.167 | 21 | 0 |
| **5** | 5 | 116.0 | 74 | 20.428571 | 26 | 25.6 | 0.201 | 30 | 0 |

In [41]:
```
df.shape
```

Out[41]:
```
(724, 9)
```

In [42]:
```
df.var()
```

Out[42]:
```
Pregnancies        10.676041
Glucose           868.450814
BloodPressure     128.189700
SkinThickness      80.621245
Insulin          6360.775382
BMI                42.783512
DPF                 0.077310
Age               137.156207
Outcome             0.223293
dtype: float64
```

In [43]:
```
df.drop('DPF',axis = 1,inplace=True) ###DPF is removed ###
```

In [44]:
```
df.Outcome.value_counts()
```

Out[44]:
```
0    481
1    243
Name: Outcome, dtype: int64
```

In [45]:
```
def sens(TP,FN):
    a=TP/(TP+FN)
    return a
def spec(TN,FP):
    b=TN/(TN+FP)
    return b
```

In [46]:
```
x=df.iloc[:,:-1].values          #iloc() used to select particular cell
of data set
y=df.iloc[:,-1].values

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.20, r
andom_state = 0)

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

```
(579, 7)
(145, 7)
(579,)
(145,)
```

#

Standard Scaler()

The standard score of a sample x is calculated as:

z = (x - u) / s

where u is the mean of the training samples or zero if with_mean=False, and s is the standard deviation of the training samples or one if with_std=False.

#

In [47]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()

x_train_std = ss.fit_transform(x_train) # fitting and transforming to no
rmal
x_test_std = ss.transform(x_test)
```

# Naive Bayes

**What is Naive Bayes algorithm?**

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

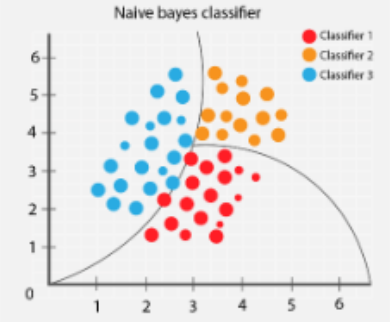
For example, some fruit may be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all these properties independently contribute to the probability that this fruit is an apple and that is why it is. known as 'Naive'.

In machine learning, Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes theorem with strong (Naive) independence assumptions between the features.

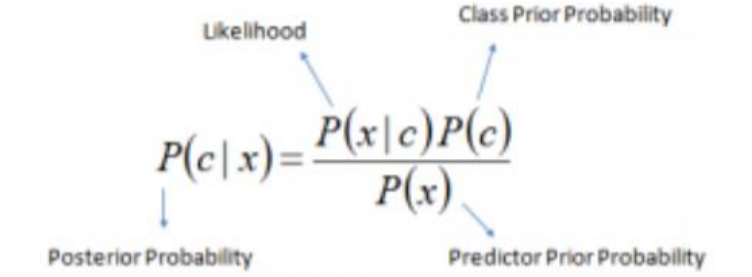
$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$




Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:




$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Above,

- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.

**Assumptions:**

The fundamental Naive Bayes assumption is that each feature makes an:

- Independent and
- Equal contribution to the outcome.

Note: The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

**How Naive Bayes algorithm works?**

Before moving to the formula for Naive Bayes, it is important to know about Bayes' theorem. Bayes' Theorem finds the probability of an event occurring given the Probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where A and B are events and P(B) ≠ 0.
Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence.
P(A) is the prior of A (the prior probability, i.e., Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event B).
P (A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.
Now, with regards to our dataset, we can apply Bayes' theorem in following way:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Where, Y is class variable and X is a dependent feature vector (of size n) where:

$$X = (x_1, x_2, x_3, ....., x_n)$$

Now, it's time to put a naive assumption to the Bayes' theorem, which is, independence among the features. So now, we split evidence into the independent parts.
Now, if any two events A and B are independent, then,

$$P(A,B) = P(A)P(B)$$

Hence, we reach to the result:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

which can be expressed as:

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

$$P(y|x_1,...,x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

Now, as the denominator remains constant for a given input, we can remove that term:

$$P(y|x_1,...,x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

Now, we need to create a classifier model. For this, we find the probability of given set of inputs for all possible values of the class variable *y* and pick up the output with maximum probability. This can be expressed mathematically as:

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

So, finally, we are left with the task of calculating P(y) and P(x, | y).
Please note that P(y) is also called **class probability** and P(x, | y) is called **conditional probability**.
The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(x, | y).

Step 1: Convert the data set into a frequency table.
Step 2: To create Likelihood table we need to find $P(x_i|y_j)$ for each $x_i$ in X and $y_j$ in Y.
Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

**Different Naive Bayes Classifiers:**

3 Popular Naive Bayes classifiers are:

The different Naive Bayes classifiers differ by the assumptions they make regarding the distribution of $P(x_i|y)$.

Binary: Binomial distribution.

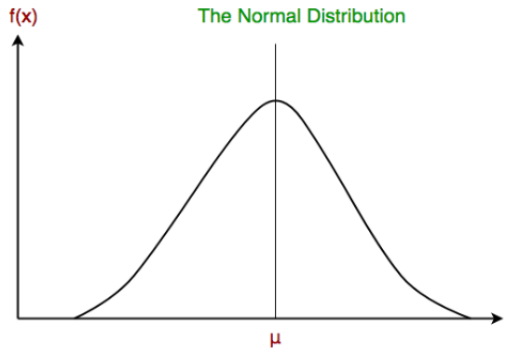
Categorical: Multinomial distribution.

Numeric: Gaussian distribution.
These three distributions are so common that the Naive Bayes implementation is often named after the distribution. For example:

Binomial Naive Bayes: If the variables are binary, such as yes/no or true/false, a binomial distribution can be used. .In Binomial Naive Bayes, values associated with each feature are assumed to be distributed according to a binomial distribution.

Multinomial Naive Bayes: In the case of categorical variables, such as counts or labels, a multinomial distribution can be used In Multinomial Naive Bayes, values associated with each feature are assumed to be distributed according to a multinomial distribution.

Gaussian Naive Bayes: If a variable is numerical, such as a measurement, often a Gaussian distribution is used. In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell-shaped curve which is symmetric about the mean of the feature values as shown below:




The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

**Advantages of Naive Bayes:**

- It is easy and fast to predict class of test data set. It also performs well in multiclass prediction.
- When assumption of independence holds, a Naive Bayes learner and classifier's perform better compare to other more sophisticated methods.
- Despite their apparently over-simplified assumptions, Naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.
- It performs well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

**Disadvantages of Naive Bayes:**

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.

- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

In [49]:
```
from sklearn.naive_bayes import GaussianNB
model_nb=GaussianNB()
model_nb.fit(x_train_std,y_train)
y_pred_nb=model_nb.predict(x_test_std)                    #predict the
 labels of the data values on the basis of the trained model
from sklearn import metrics                               # metrics lib
=measure classification performance
accugaussNB= metrics.accuracy_score (y_test,y_pred_nb)    #computes acc
uracy
print("Accuracy for Gaussian Naive Bayes is", accugaussNB)
```

```
Accuracy for Gaussian Naive Bayes is 0.7931034482758621
```

In [50]:
```
print(metrics.classification_report(y_test,y_pred_nb))
print(metrics.confusion_matrix(y_test,y_pred_nb))
```

```
              precision    recall  f1-score   support

           0       0.84      0.87      0.86       103
           1       0.66      0.60      0.62        42

    accuracy                           0.79       145
   macro avg       0.75      0.73      0.74       145
weighted avg       0.79      0.79      0.79       145

[[90 13]
 [17 25]]
```

In [51]:
```
TP,TN,FP,FN=90,25,17,13
anb=sens(TP,FN)
bnb=spec(TN,FP)
print("Sensitivity is ",anb)
print("Specificity is ",bnb)
```

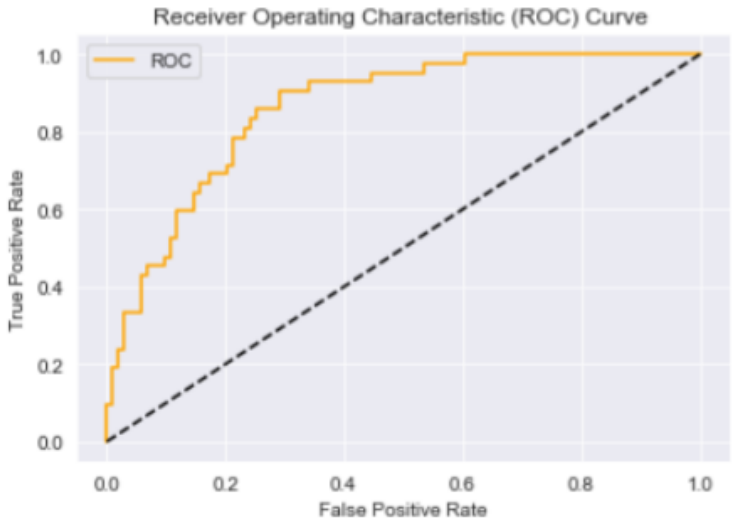
```
Sensitivity is  0.8737864077669902
Specificity is  0.5952380952380952
```

In [52]:
```
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
probas_pred_nb=model_nb.predict_proba(x_test_std)
fpnb, tpnb, thresholdsnb = roc_curve(y_test,probas_pred_nb[:,1],pos_labe
l=1)
roc_auc_nb=auc(fpnb, tpnb)
print("AUC for Gaussian Naive Bayes : ",roc_auc_nb)
```

```
AUC for Gaussian Naive Bayes :  0.8606102635228849
```

```
In [53]: def plot_roc_curve(fpnb,tpnb):
             plt.plot(fpnb,tpnb,color='orange',label='ROC')
             plt.plot([0,1],[0,1],color='black',linestyle='--')
             plt.xlabel('False Positive Rate')
             plt.ylabel('True Positive Rate')
             plt.title('Receiver Operating Characteristic (ROC) Curve ')
             plt.legend()
             plt.show
```

```
In [54]: plot_roc_curve(fpnb,tpnb)
```




# K-Nearest Neighbours (K-NN)

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

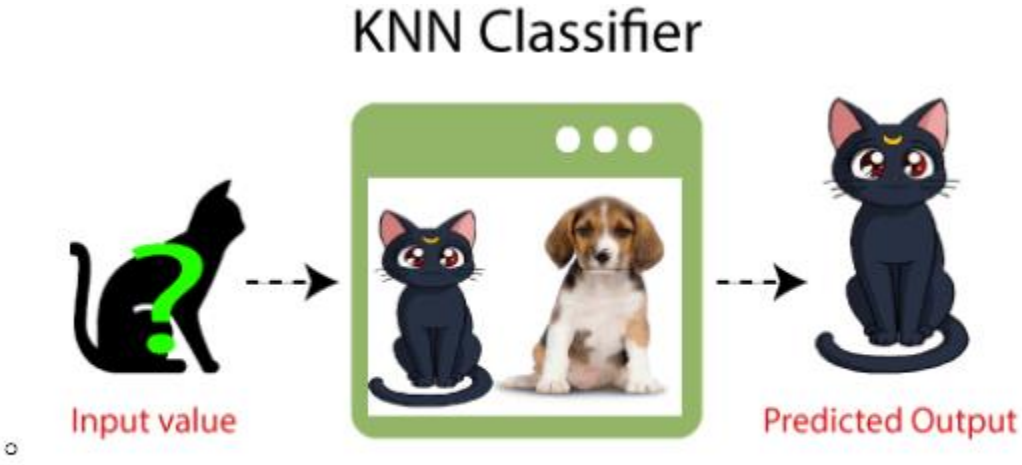
K-NN algorithm assumes the similarity between the new case/data and available cases and puts the new case into the available category that shows the most similarity.

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.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

**Example**: Suppose, we have an image of a creature that looks like cat and dog, but we want to know either it is a cat or dog. So, for this identification, we can use the K-NN algorithm, as it works on a similarity measure. Our K-NN model will find the similar features of the new

data set to the cats' and dogs' images and based on the most similar features it will put it in either cat or dog category.




**Features of K-NN Algorithm:**

Supervised Learning algorithmic

Supervised learning is based on training data it labels and memorize the data into classified sets. Here in our data if we have 1, it is classified as diabetic patient and for 0, it is classified as healthy person.

Simple

It works on amazingly simple concept, when you put in the new data it's going to check to which neighbour you are like, from two types of neighbour.

Non-Parametric

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

Lazy Algorithm

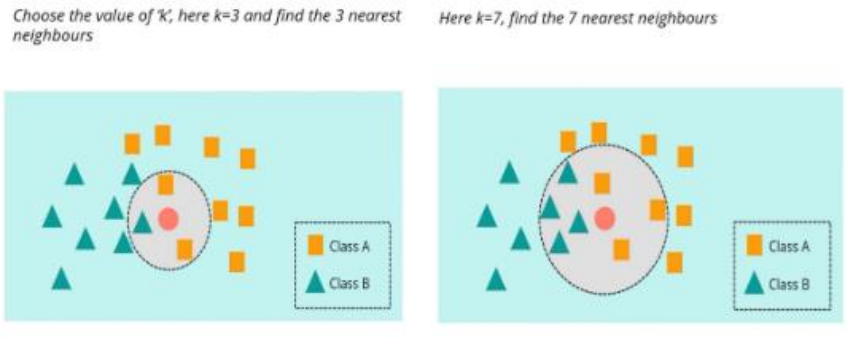
Here, Lazy means it does not learn the data it just memorizes the data, unlike other parametric models where in you learn values, learn different predictions.

Feature similarity

K-NN is based on feature similarity i.e., it is going to look around and look at the feature and check how similar I am as compared to my neighbours.

**How does K-NN algorithm works?**

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. We use Euclidean distance formula to calculate distance between two points. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:




The K-NN working can be explained based on the below algorithm:

Step-1: Select the number K of the neighbours.

Step-2: Calculate the Euclidean distance of K number of neighbours.

Step-3: Take the K nearest neighbours as per the calculated Euclidean distance.

Step-4: Among these K neighbours, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbour is maximum.

Step-6: Our model is ready.

**How to select the value of K in the K-NN Algorithm?**

To build the model, value of K needs to be chosen. Having K as an odd number ensures there are no ties, hence making the algorithm easier to implement. We perform 10-fold cross validation to pick one among 20 values of K. Note that for K-NN, the data must be normalized, i.e., centered and scaled.

Below are some points to remember while selecting the value of K in the K-NN algorithm:

There is no particular way to determine the best value for K; so we need to try some values to find the best out of them. The most preferred value for K is the value nearest to the square root of the number of observations.

An extremely low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.

**Advantages of K-NN Algorithm:**

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

**Disadvantages of K-NN Algorithm:**

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

```
In [55]: ##########KNN Classifier#############

In [56]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn import metrics
         #how the performance of machine learning algorithms is measured
         from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
         # grid= loop through predefined hyperparameters
         knn=KNeighborsClassifier()
         param_grid = {'n_neighbors':[5,10,15,25,30,50]}
         grid_knn = GridSearchCV(knn,param_grid,scoring='accuracy',cv = 10,refit
         = True)
         grid_knn.fit(x_train_std,y_train)

Out[56]: GridSearchCV(cv=10, error_score=nan,
                      estimator=KNeighborsClassifier(algorithm='auto', leaf_size=
         30,
                                                     metric='minkowski',
                                                     metric_params=None, n_jobs=N
         one,
                                                     n_neighbors=5, p=2,
                                                     weights='uniform'),
                      iid='deprecated', n_jobs=None,
                      param_grid={'n_neighbors': [5, 10, 15, 25, 30, 50]},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=Fal
         se,
                      scoring='accuracy', verbose=0)
```

In [57]:
```
# minkowski distance formula
# deprecated= if code is faulty it replaces with new code for diff value
of k
# paramgrid=checking for different values of k
# pre_dispatch = task performed in queue
# verbose= produce detailed output
# cv= cross validation
# p=2 == we are using euclidean distance.
```

In [58]:
```
print("Tuned Parameters==>",grid_knn.best_params_)
```

```
Tuned Parameters==> {'n_neighbors': 30}
```

In [59]:
```
knn = KNeighborsClassifier(n_neighbors=30)
knn.fit(x_train_std,y_train)
y_pred_knn=knn.predict(x_test_std)
from sklearn import metrics
accuKNN=metrics.accuracy_score(y_test,y_pred_knn)
print("Accuracy for KNN Classifier is",accuKNN)
```

```
Accuracy for KNN Classifier is 0.8275862068965517
```

In [60]:
```
print(metrics.classification_report(y_test,y_pred_knn))
print(metrics.confusion_matrix(y_test,y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.83      0.95      0.89       103
           1       0.81      0.52      0.64        42

    accuracy                           0.83       145
   macro avg       0.82      0.74      0.76       145
weighted avg       0.83      0.83      0.81       145

[[98  5]
 [20 22]]
```

In [61]:
```
TP,TN,FP,FN=98,22,20,5
aknn=sens(TP,FN)
bknn=spec(TN,FP)
print("Sensitivity is ",aknn)
print("Specificity is ",bknn)
```

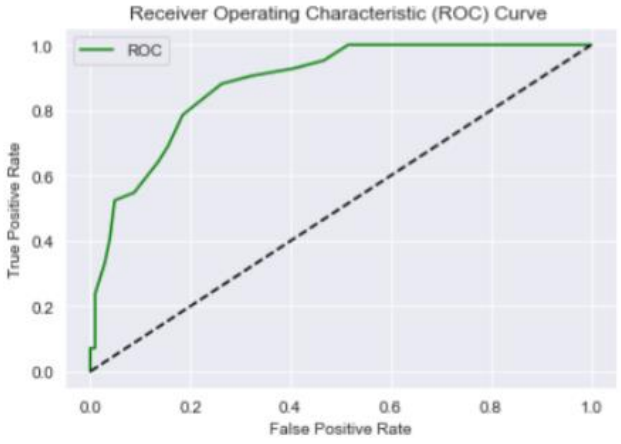
```
Sensitivity is  0.9514563106796117
Specificity is  0.5238095238095238
```

In [62]:
```
probas_pred_knn=knn.predict_proba(x_test_std)
fpknn, tpknn, thresholdsknn = roc_curve(y_test,probas_pred_knn[:,1],pos_
label=1)
roc_auc_knn=auc(fpknn, tpknn)
print("AUC for K Nearest Neighbors : ",roc_auc_knn)
```

```
AUC for K Nearest Neighbors :  0.8829172445677301
```

In [63]:
```
def plot_roc_curve(fpknn,tpknn):
    plt.plot(fpknn,tpknn,color='green',label='ROC')
    plt.plot([0,1],[0,1],color='black',linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve ')
    plt.legend()
    plt.show
```

In [64]:
```
plot_roc_curve(fpknn,tpknn)
```



# Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve complex problem and to improve the performance of the model. As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of over fitting.

**Why Random Forest and not Decision Tree?**

While random forest is a collection of decision trees, there are some differences or example, to predict whether a person will click on an online advertisement, you might collect the ads the person clicked on in the past and some features that describe his/her decision. If you put the features and labels into a decision tree, it will generate some rules that help predict whether the advertisement will be clicked or not. In comparison, the random forest algorithm randomly selects observations and features to build several decision trees and then averages the results. Another difference is "deep" decision trees might suffer from over fitting. Most of the time, random forest prevents this by creating random subsets of the features and building smaller trees using those subsets. Afterwards, it combines the sub-trees. It's important to note this doesn’t work every time and it also makes the computation slower, depending on how many trees the random forest builds.

**Assumptions:**

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random Forest Classifier:

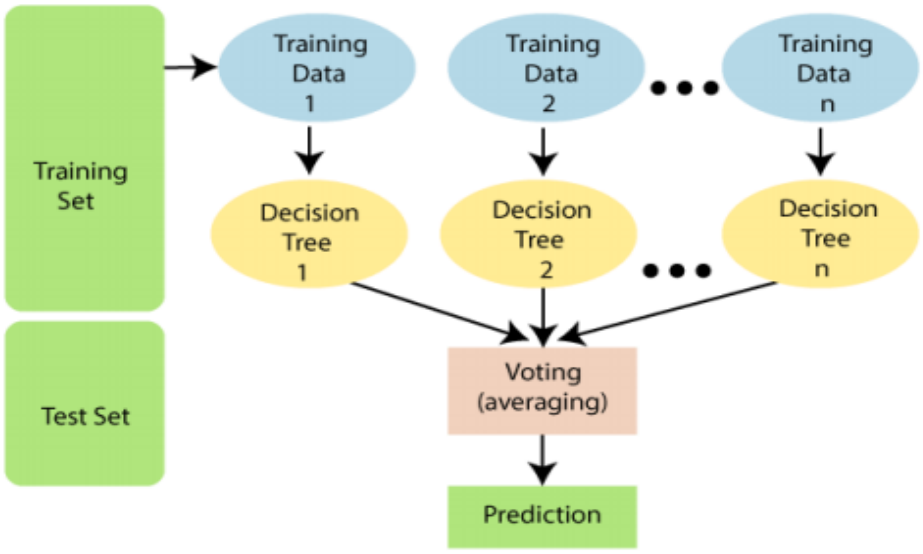
There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.

The predictions from each tree must have extremely low correlations.

**Working of Random Forest:**

Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

The below diagram explains the working of the Random Forest algorithm:




One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems.

In Random Forest each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random but with replacement, from the original data. This sample will be the training set for growing there.

2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.

3. Each tree is grown to the largest extent possible. There is no pruning.

**Advantages of Random Forest:**

- It can be used for both regression and classification tasks.
- It is also easy to view the relative importance it assigns to the input features.
- Random forest is also a very handy algorithm.

**Disadvantages of Random Forest:**

- The main limitation of random forest is that many trees can make the algorithm too slow and ineffective for real-time predictions.
- A more accurate prediction requires more trees, which results in as slower model.

In [66]:
```
########Random Forest Classifier########
```

In [67]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
param_grid={'n_estimators':[200,500,1000]}
grid_rfc=RandomizedSearchCV(rfc,param_grid,n_iter=20,scoring='accuracy',
cv=10,refit=True)
grid_rfc.fit(x_train_std,y_train)
```

Out[67]:
```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decreas
e=0.0,
                                                    min_impurity_split=N
one,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_
leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs=None,
                                                    oob_score=False,
                                                    random_state=None,
                                                    verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=20, n_jobs=None,
                   param_distributions={'n_estimators': [200, 500, 100
0]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=Tru

e,
                   return_train_score=False, scoring='accuracy', verbose
=0)
```

In [68]:
```
# n_iter= (number of iterations) or (trial and score)
# RandomizedSearchCV implements a “fit”, “score” method,predict,etc,etc
```

In [69]:
```
print("Tuned Parameters==>",grid_rfc.best_params_)
```

```
Tuned Parameters==> {'n_estimators': 200}
```

In [70]:
```
rfc=RandomForestClassifier(n_estimators=200)
rfc.fit(x_train_std,y_train)
```

Out[70]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='a
uto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=Non
e,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [71]:
```
y_pred_rfc=rfc.predict(x_test_std)
from sklearn import metrics
accuRF=metrics.accuracy_score(y_test,y_pred_rfc)
print("Accuracy for Random Forest Classifier is",accuRF)
```

```
Accuracy for Random Forest Classifier is 0.7931034482758621
```

In [72]:
```
print(metrics.classification_report(y_test,y_pred_rfc))
print(metrics.confusion_matrix(y_test,y_pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       103
           1       0.69      0.52      0.59        42

    accuracy                           0.79       145
   macro avg       0.76      0.71      0.73       145
weighted avg       0.78      0.79      0.78       145

[[93 10]
 [20 22]]
```

In [73]:
```
TP,TN,FP,FN=93,22,20,10
arf=sens(TP,FN)
brf=spec(TN,FP)
print("Sensitivity is ",arf)
print("Specificity is ",brf)
```

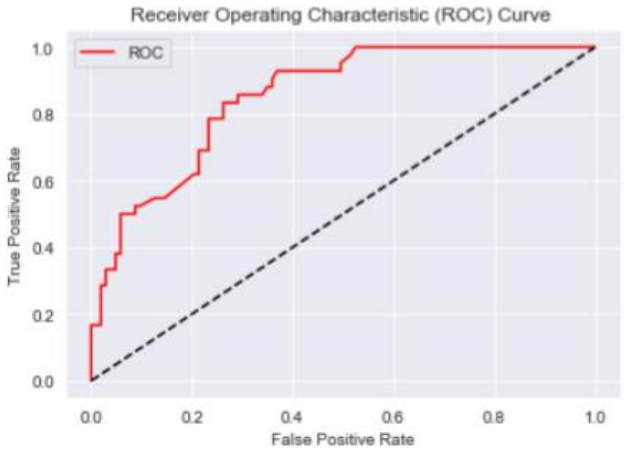
```
Sensitivity is  0.9029126213592233
Specificity is  0.5238095238095238
```

In [74]:
```
probas_pred_rfc=rfc.predict_proba(x_test_std)
fprf, tprf, thresholdsrf = roc_curve(y_test,probas_pred_rfc[:,1],pos_lab
el=1)
roc_auc_rf=auc(fprf, tprf)
print("AUC for Random Forest : ",roc_auc_rf)
```

```
AUC for Random Forest :  0.8513638465094776
```

```
In [75]: def plot_roc_curve(fprf,tprf):
             plt.plot(fprf,tprf,color='red',label='ROC')
             plt.plot([0,1],[0,1],color='black',linestyle='--')
             plt.xlabel('False Positive Rate')
             plt.ylabel('True Positive Rate')
             plt.title('Receiver Operating Characteristic (ROC) Curve ')
             plt.legend()
             plt.show
```

```
In [76]: plot_roc_curve(fprf,tprf)
```




# Logistic Regression

**What is Logistic Regression?**

Logistic regression is the regression which produces results in binary format and is used to predict the outcome of a categorical dependent variable. So, the outcome should be discrete/categorical for example, 0 or 1, yes or no, true or false, etc.

We must model P(X) using a function that gives outputs between 0 and 1 for all values of X. In logistic regression, we use the **logistic function.**

$$P(X) = \frac{e^{\beta o + \beta 1 x}}{1 + e^{\beta o + \beta 1 x}}$$

i.e., Y=P(x)+∈ …… (1)

βo, $\beta_1$ are the regression coefficients. A transformation of P(x) that is useful in the study of logistic regression is logit transformation.

**Logit transformation**=ln(P(X)/(1-P(X)) …… (2)

We see that the logistic regression model has a logit that is linear in X. Fitting of logistic regression model is done by maximum likelihood estimation.

The logistic function will always produce an S-shaped curve of this form, and so regardless of the value of X, we will obtain a sensible prediction. We also see that the logistic model is better able to capture the range of probabilities. We find that,

$P(X)/ [1 - P(X)] = e^{\beta 0+\beta 1x}$ …….…… (3)

The quantity P(X)/[1−P(X)] is called the **odds** and can take on any value odds between 0 and ∞. In a logistic regression model, increasing X by one unit changes the log odds by $\beta_1$ or equivalently it multiplies the odds by $e^{\beta 1}$.

As the relationship between P(X) and X in is not a straight line, $\beta_1$ does not correspond to the change in P(X) associated with a unit increase in X. The amount that P(X) changes due to a one-unit change in X will depend on the current value of X. But regardless of the value of X, if $\beta_1$ is positive then increasing X will associated with increasing P(X), and if $\beta_1$ is negative then increasing X will be associated with decreasing P(X).

Likelihood estimation

Method of maximum likelihood is used since it has better statistical properties. The basic idea behind using maximum likelihood to fit a logistic regression model is as follow we try to find $\beta_o$ and $\beta_1$ such that plugging these estimates into the model for P(X) gives a number close to one for all individuals who have diabetes and a number close to zero for all individuals who did not. This can be formalized using a mathematical equation called a likelihood function,

$L (\beta o, \beta_1) = \prod_i^n P(x_i) * \prod_i^n [1 - P(x_i)]$

The estimates βo and $\beta_1$ are chosen to maximize this likelihood function.

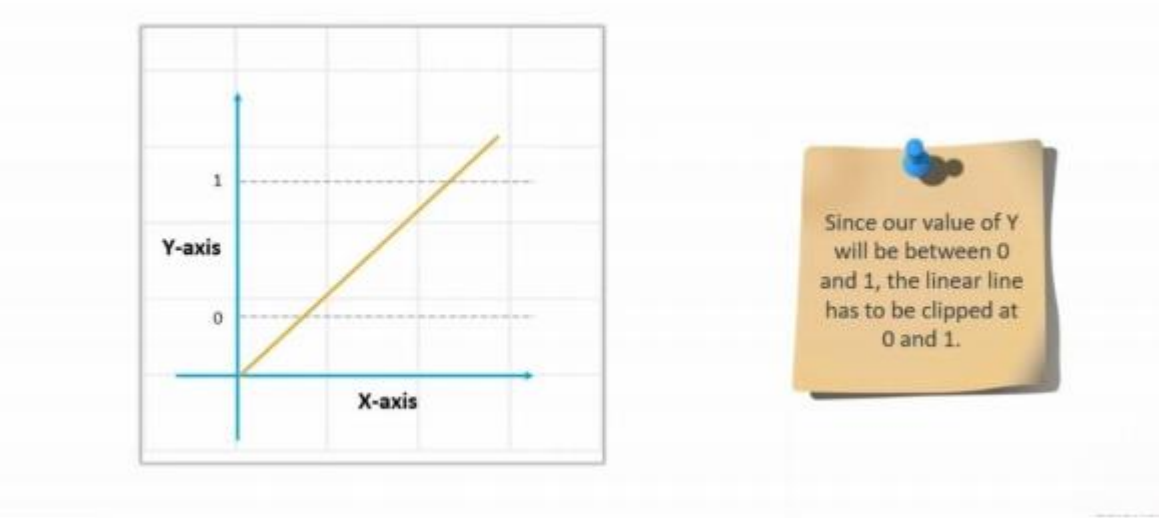
We fit a logistic model to the train data.

**Assumptions:**

Logistic regression does not make many of the key assumptions of linear regression and general linear models that are based on ordinary least squares algorithms – particularly regarding linearity, normality, homoscedasticity, and measurement level.
However, some other assumptions still apply.
First, binary logistic regression requires the dependent variable to be binary.
Second, logistic regression requires the observations to be independent of each other.
Third, the independent variables should not be too highly correlated with each other.
Fourth, logistic regression assumes linearity of independent variables and log odds
Finally, logistic regression typically requires a large sample size.

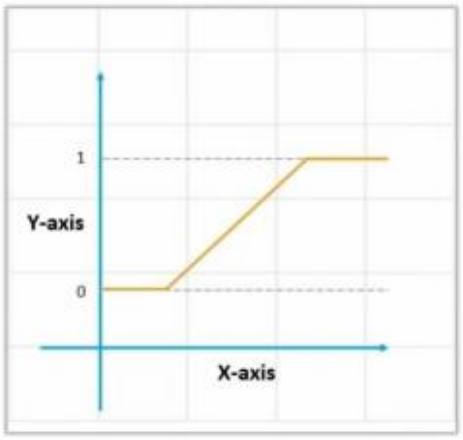
**Why logistic and not linear?**

In linear regression, the dependent variable ranges from depending upon values of independent variables which doesn't provides us actual view of categorical classification. But in logistic regression, as the dependent variable can only bear discrete binary values (0 or1), it gives the broader view of categorical classification and helps us in prediction.



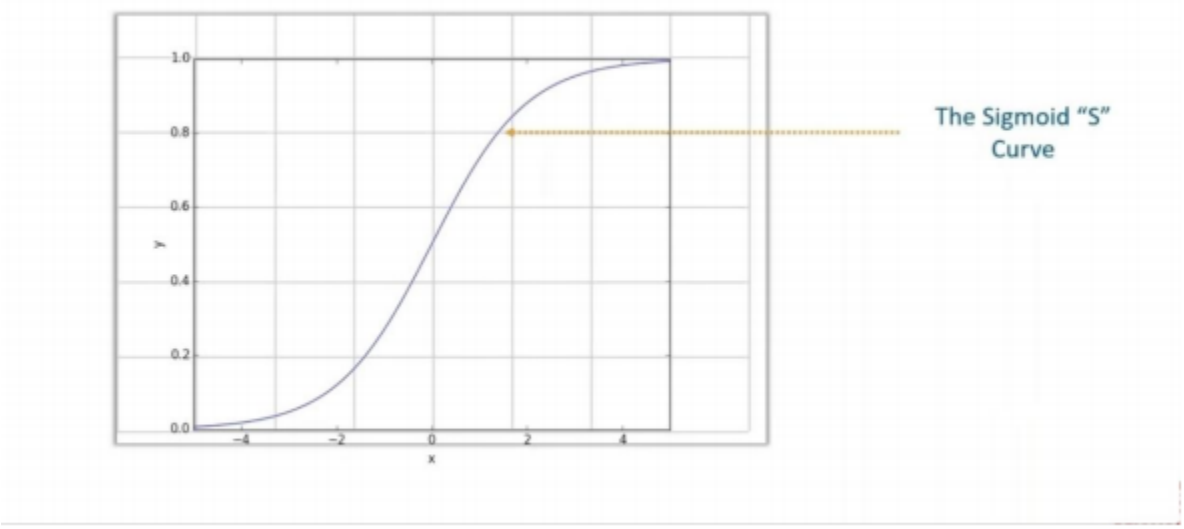

The above graph provides us the straight-line curve of linear regression in which the dependent variable Y could take values ranging from depending upon values of X. But our motive is to obtain outcome in binary format; hence we have to clipped values at 0 and 1.

The next graph gives us idea how graph looks after clipping at 0 and 1.



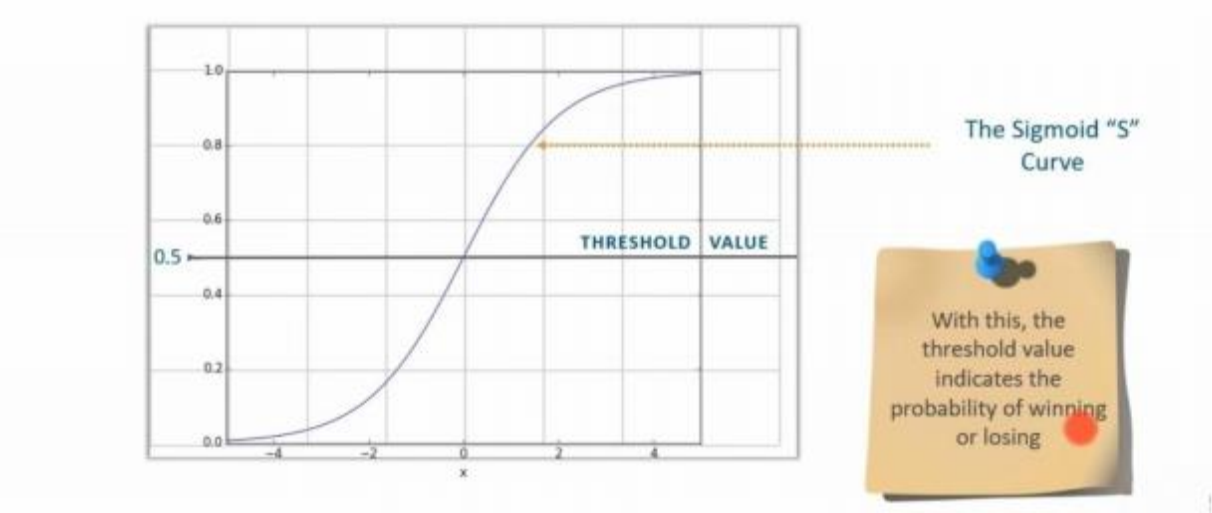


Now, keeping this curve in mind we came up with the sigmoid (S) curve which represents logistic regression. The sigmoid curve is given as




**How Logistic works?**

But one cannot decide the outcome as 0 or 1 by means of probabilities in decimals. Hence, we've to decide threshold for above curve which will give us the clear classification between outcomes. Threshold probability acts as a barrier which classifies the outcomes in into winning (1) or losing (0) categories. Considering threshold as 0.5 our graph looks like,




**Advantages of Logistic Regression:**

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

**Disadvantages of Logistic Regression:**

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

In [76]:
```
#######Logistic Regression######
```

In [77]:
```
from sklearn.linear_model import LogisticRegression
model_lr= LogisticRegression()
model_lr.fit(x_train_std,y_train)
y_pred_lr=model_lr.predict(x_test_std)
from sklearn import metrics
accuLR=metrics.accuracy_score(y_test,y_pred_lr)
print('The Accuracy of the Logistic Regression is',accuLR)
```

```
The Accuracy of the Logistic Regression is 0.8137931034482758
```

In [78]:
```
print(metrics.classification_report(y_test,y_pred_lr))
print(metrics.confusion_matrix(y_test,y_pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.84      0.91      0.87       103
           1       0.73      0.57      0.64        42

    accuracy                           0.81       145
   macro avg       0.78      0.74      0.76       145
weighted avg       0.81      0.81      0.81       145

[[94  9]
 [18 24]]
```

In [79]:
```
TP,TN,FP,FN=94,24,18,9
alr=sens(TP,FN)
blr=spec(TN,FP)
print("Sensitivity is ",alr)
print("Specificity is ",blr)
```

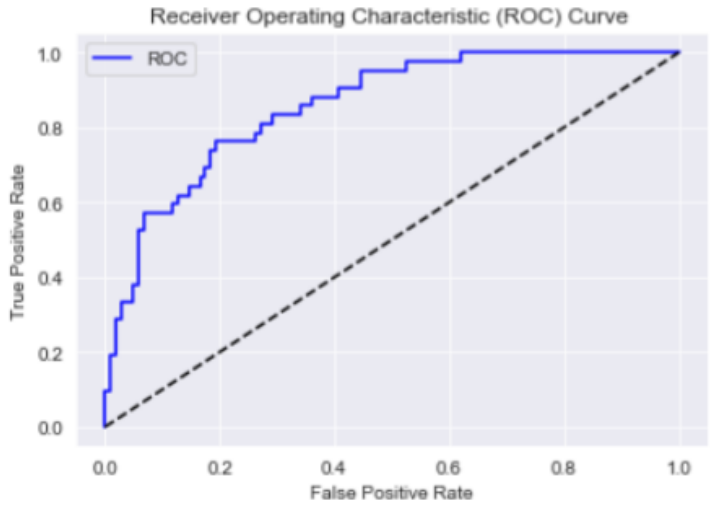
```
Sensitivity is  0.912621359223301
Specificity is  0.5714285714285714
```

In [80]:
```
probas_pred_lr=model_lr.predict_proba(x_test_std)
fplr, tplr, thresholdslr = roc_curve(y_test,probas_pred_lr[:,1],pos_labe
l=1)
roc_auc_lr=auc(fplr, tplr)
print("AUC for Logistic Regression Model : ",roc_auc_lr)
```

```
AUC for Logistic Regression Model :  0.8566805362921868
```

In [81]:
```
def plot_roc_curve(fplr,tplr):
    plt.plot(fplr,tplr,color='blue',label='ROC')
    plt.plot([0,1],[0,1],color='black',linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve ')
    plt.legend()
    plt.show
```

```
In [82]: plot_roc_curve(fplr,tplr)
```




# Decision Trees

**What is Decision Tree?**

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. Classification and Regression Trees or CART for short is a term introduced by Leo Bier man to refer to Decision Tree algorithms that can be used for classification or regression predictive modelling problems. Classically, this algorithm is referred to as “decision trees”, but on some platforms like R they are referred to by the more modern term CART. The CART algorithm provides a foundation for important algorithms like bagged decision trees, random forest and boosted decision trees.

**Types of Decision Trees:**

There are two main types of decision trees that are based on the target variable, i.e., classification tree and regression tree.

1. Classification Tree

A Classification tree is built through a process known as binary recursive partitioning. This is an iterative process of splitting the data into partitions, and then splitting it up further on each of the branches.

Classification Trees are used when the dependent variable is categorical or qualitative (e.g. if we want to estimate the blood type of a person).

2. Regression Tree

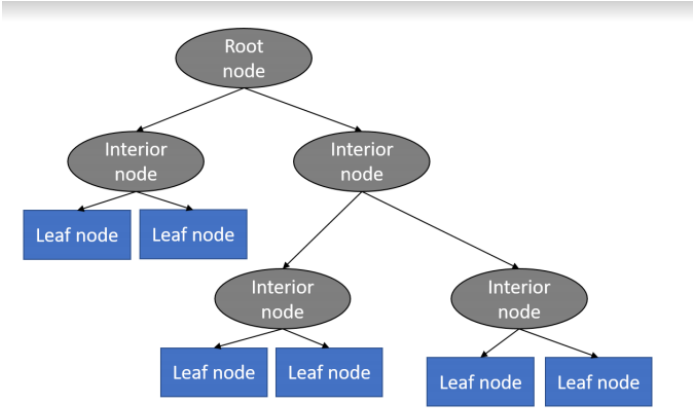
A regression tree is also built through a process known as binary recursive partitioning, which is an iterative process that splits the data into partitions or branches, and then continues splitting each partition into smaller groups as the method moves up each branch.

Regression Trees are used when the dependent variable is continuous or quantitative (e.g. if we want to estimate the probability that a customer will default on a loan)

The primary difference between classification and regression decision trees is that, the classification decision trees are built with unordered values with dependent variables. The regression decision trees take ordered values with continuous values.

**Terminology of Regression Trees:**

As stated, earlier Decision tree is a supervised learning algorithm. It classifies instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. It is tree like structure starts with the root node at the top, the root node, where the process of decision-making starts. Under the Root nodes are Interior nodes or simply known as decision nodes. They can have two or more branches as per needed. Lastly comes the Leaf nodes which represents the classification or decision of the instance. This node has no additional nodes coming off them.




**Root:** This is the beginning of decision tree, which also represents the population sample. For example, say you want to decide the best performing employee in an organization based on various criteria, such as attendance of the employee, number of successful projects, number of employees he/she mentored, etc. So here, the entire population of employees is at the root of the decision tree.
**Leaf:** The terminal node is called the leaf node. In our example, the final best employee would be the leaf node or the terminal node.
**Decision Node:** Here the other nodes are divided into the further categories. In our example, the various criteria would determine a decision node.

**Child Node:** When a node is divided into other subparts, subparts are called child nodes. And the node which is divided is called the **parent node.**

**Construction of Decision Tree:**

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

**Advantages of Decision Trees:**

- A user can visualize each step, which can help with making rational decisions.
- You can give the priority to a decision criterion. For example, in our employee example, you can put the attendance criteria on the top of the decision tree if that is the most important criteria.
- Making a decision based on regression is much easier than most other methods. Since most of the undesired data will be filtered outlier each step, you must work on less data as you go further in the tree.
- It is easy to prepare a regression tree. A user can present it to the higher authorities in a much easier way as it can be represented on a simple chart or diagram.

**Disadvantages of Decision trees:**

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.
- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used, and a search must be. made for optimal combining weights. Pruning algorithms can also be. expensive since many candidates sub-trees must be formed and compared.

In [84]:
```
##############Decision Tress###########
```

In [85]:
```
from sklearn.tree import DecisionTreeClassifier
model_dt=DecisionTreeClassifier()
model_dt.fit(x_train_std,y_train)
y_pred_dt=model_dt.predict(x_test_std)
from sklearn import metrics
accuDT=metrics.accuracy_score(y_test,y_pred_dt)
print('The Accuracy of Decision tree model is',accuDT)
```

```
The Accuracy of Decision tree model is 0.7241379310344828
```

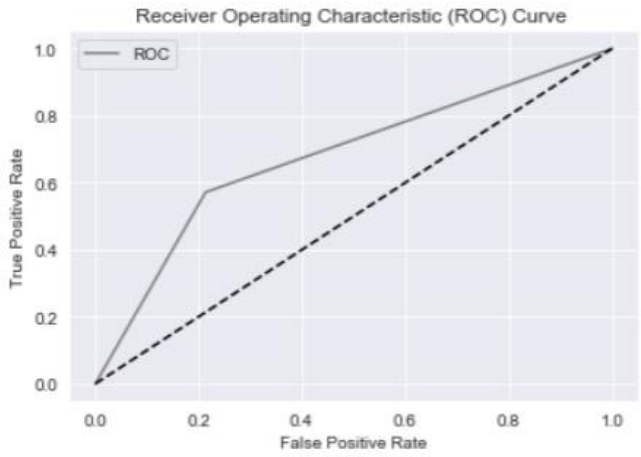
In [86]:
```
test_pred_dt=model_dt.predict(x_test_std)
print(metrics.classification_report(y_test,test_pred_dt))
print(metrics.confusion_matrix(y_test,test_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.82      0.79      0.80       103
           1       0.52      0.57      0.55        42

    accuracy                           0.72       145
   macro avg       0.67      0.68      0.67       145
weighted avg       0.73      0.72      0.73       145

[[81 22]
 [18 24]]
```

In [87]:
```
TP,TN,FP,FN=81,24,18,22
adt=sens(TP,FN)
bdt=spec(TN,FP)
print("Sensitivity is ",adt)
print("Specificity is ",bdt)
```

```
Sensitivity is  0.7864077669902912
Specificity is  0.5714285714285714
```

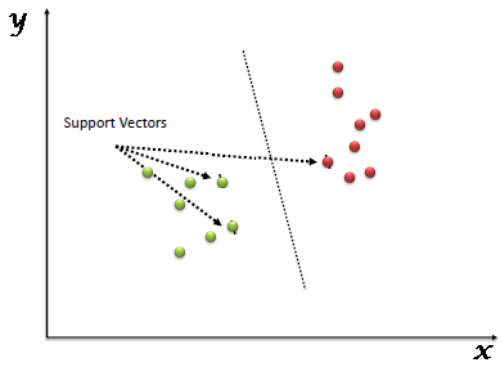
```
In [88]: probas_pred_dt=model_dt.predict_proba(x_test_std)
         fpdt, tpdt, thresholdsdt = roc_curve(y_test,probas_pred_dt[:,1],pos_labe
         l=1)
         roc_auc_dt=auc(fpdt, tpdt)
         print("AUC for Decision Trees : ",roc_auc_dt)

         AUC for Decision Trees :  0.6789181692094313

In [89]: def plot_roc_curve(fpdt,tpdt):
             plt.plot(fpdt,tpdt,color='grey',label='ROC')
             plt.plot([0,1],[0,1],color='black',linestyle='--')
             plt.xlabel('False Positive Rate')
             plt.ylabel('True Positive Rate')
             plt.title('Receiver Operating Characteristic (ROC) Curve ')
             plt.legend()
             plt.show

In [90]: plot_roc_curve(fpdt,tpdt)
```




# Support Vector Machine

**What is Support Vector Machine?**

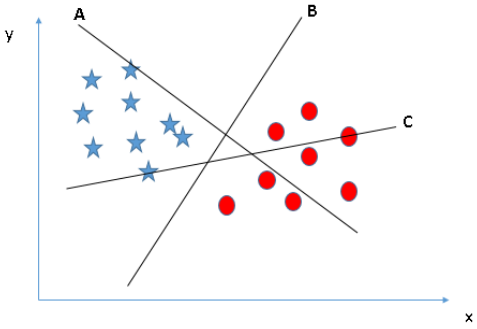
"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification and regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features 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 (look at the below snapshot). Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).




**How does it work?**

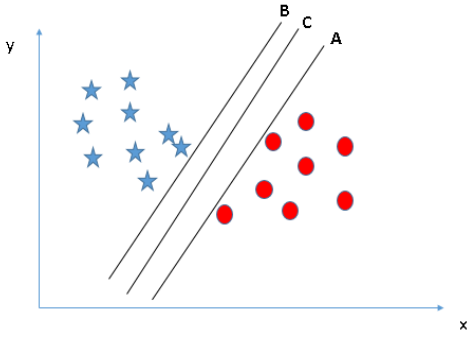
Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is "How can we identify the right hyper-plane?". Don't worry, it's not as hard as you think!

Let's understand:

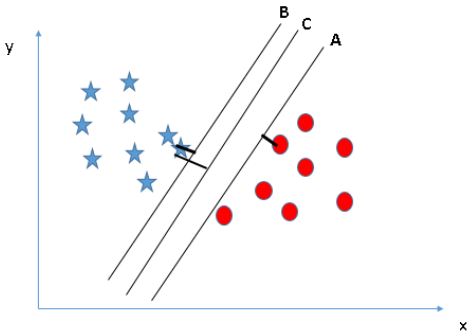



**Identify the right hyper-plane (Scenario-1):** Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.

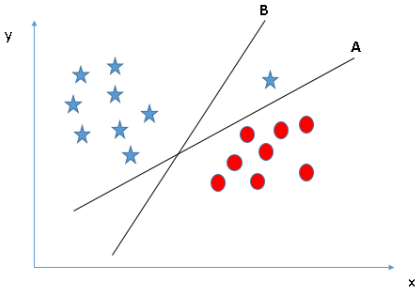
You need to remember a thumb rule to identify the right hyper-plane: "Select the hyper-plane which segregates the two classes better". In this scenario, hyper-plane "B" has excellently performed this job.



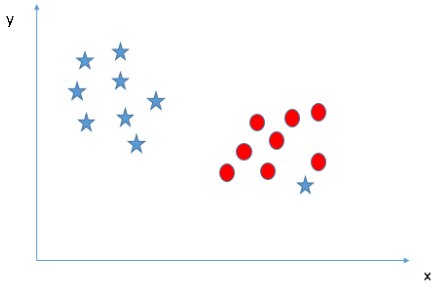

**Identify the right hyper-plane (Scenario-2):** Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, how can we identify the right hyper-plane?



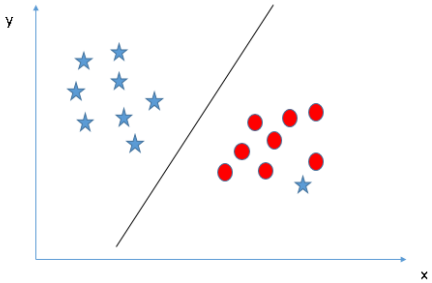

Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Let's look at the snapshot: Here, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.



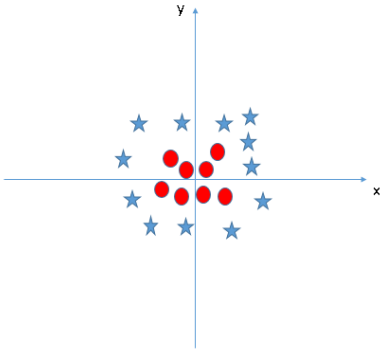

**Identify the right hyper-plane (Scenario-3):** Hint: Use the rules as discussed in previous section to identify the right hyper-plane. Some of you may have selected the hyper-plane **B** as it has higher margin compared to **A.** But here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**



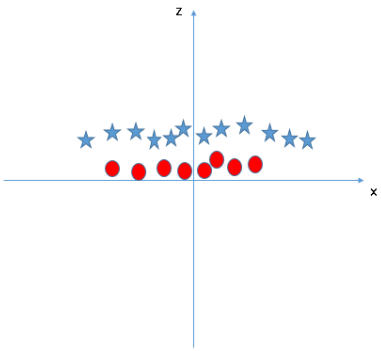

**Can we classify two classes (Scenario-4)? -**Below, We are unable to segregate the two classes using a straight line, as one of the stars lies in the territory of other (circle) class as an outlier.




As mentioned, one star at another end is like an outlier for star class. The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin. Hence, we can say, SVM classification is robust to outliers.




**Find the hyper-plane to segregate to classes (Scenario-5):** In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane. SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature $z=x^2+y^2$. Now, let's plot the data points on axis x and z:

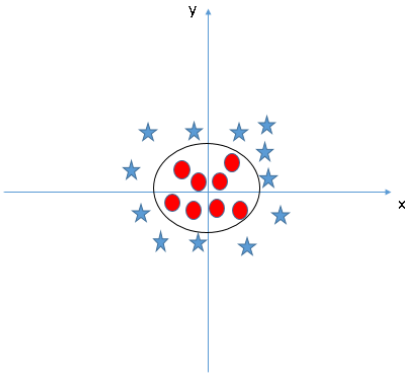



In beside plot, points to consider are:

All values for z would be positive always because z is the squared sum of both x and y

In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

In the SVM classifier, it is easy to have a linear hyper-plane between these two classes. But another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, the SVM algorithm has a technique called the kernel trick. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e., it converts not separable problem to separable problem. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined. When we look at the hyper-plane in original input space it looks like a circle:



**Advantages of SVM:**

- It works really well with a clear margin of separation
- It is effective in high dimensional spaces.
- It is effective in cases where the number of dimensions is greater than the number of samples.
- It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

**Disadvantages of SVM:**

- It doesn't perform well when we have large data set because the required training time is higher
- It also doesn't perform very well, when the data set has more noise i.e., target classes are overlapping
- SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is included in the related SVC method of Python scikit-learn library.

```
In [90]: ##########SVC-linear#############
```

```
In [91]: from sklearn.svm import SVC
svc_linear= SVC(C= 0.5, kernel= 'linear', random_state= 0, probability=T
rue)
svc_linear.fit(x_train_std, y_train)
y_pred_svc_linear=svc_linear.predict(x_test_std)
from sklearn import metrics
accuSVC_linear=metrics.accuracy_score(y_test,y_pred_svc_linear)
print("Accuracy for SVC-linear Classifier is",accuSVC_linear)
```

```
Accuracy for SVC-linear Classifier is 0.8137931034482758
```

```
In [92]: print(metrics.classification_report(y_test,y_pred_svc_linear))
print(metrics.confusion_matrix(y_test,y_pred_svc_linear))
```

```
              precision    recall  f1-score   support

           0       0.84      0.91      0.87       103
           1       0.73      0.57      0.64        42

    accuracy                           0.81       145
   macro avg       0.78      0.74      0.76       145
weighted avg       0.81      0.81      0.81       145

[[94  9]
 [18 24]]
```

In [93]:
```
TP,TN,FP,FN=94,24,18,9
asvclinear=sens(TP,FN)
bsvclinear=spec(TN,FP)
print("Sensitivity is ",asvclinear)
print("Specificity is ",bsvclinear)
```

```
Sensitivity is  0.912621359223301
Specificity is  0.5714285714285714
```

In [94]:
```
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
probas_pred_svc_linear=svc_linear.predict_proba(x_test_std)
fpsvclinear, tpsvclinear, thresholdssvclinear = roc_curve(y_test,probas_
pred_svc_linear[:,1],pos_label=1)
roc_auc_svc_linear=auc(fpsvclinear, tpsvclinear)
print("AUC for SVC-linear : ",roc_auc_svc_linear)
```

```
AUC for SVC-linear :  0.858067498844198
```

In [95]:
```
def plot_roc_curve(fpsvclinear,tpsvclinear):
    plt.plot(fpsvclinear,tpsvclinear,color='cyan',label='ROC')
    plt.plot([0,1],[0,1],color='black',linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve ')
    plt.legend()
    plt.show
```

In [96]:
```
plot_roc_curve(fpsvclinear,tpsvclinear)
```


In [97]:
```
##########SVC-radial###########
```

In [98]:
```
from sklearn.svm import SVC
svc_radial= SVC(C=0.5, kernel= 'rbf',random_state= 0, gamma=0.12,tol=0.0
0001, probability=True)
svc_radial.fit(x_train_std, y_train)
y_pred_svc_radial=svc_radial.predict(x_test_std)
from sklearn import metrics
accuSVC_radial=metrics.accuracy_score(y_test,y_pred_svc_radial)
print("Accuracy for SVC-radial Classifier is",accuSVC_radial)
```

```
Accuracy for SVC-radial Classifier is 0.8068965517241379
```

In [99]:
```
print(metrics.classification_report(y_test,y_pred_svc_radial))
print(metrics.confusion_matrix(y_test,y_pred_svc_radial))
```

```
              precision    recall  f1-score   support

           0       0.82      0.93      0.87       103
           1       0.75      0.50      0.60        42

    accuracy                           0.81       145
   macro avg       0.79      0.72      0.74       145
weighted avg       0.80      0.81      0.79       145

[[96  7]
 [21 21]]
```

In [100]:
```
TP,TN,FP,FN=96,21,21,7
asvcradial=sens(TP,FN)
bsvcradial=spec(TN,FP)
print("Sensitivity is ",asvcradial)
print("Specificity is ",bsvcradial)
```

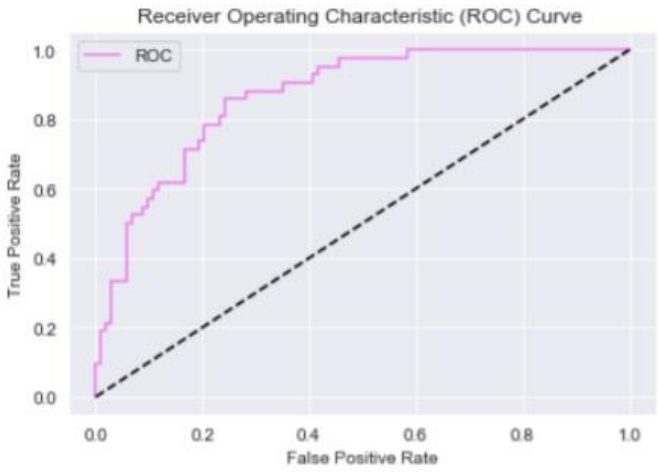
```
Sensitivity is  0.9320388349514563
Specificity is  0.5
```

In [101]:
```
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
probas_pred_svc_radial=svc_radial.predict_proba(x_test_std)
fpsvcradial, tpsvcradial, thresholdssvcradial = roc_curve(y_test,probas_
pred_svc_radial[:,1],pos_label=1)
roc_auc_svc_radial=auc(fpsvcradial, tpsvcradial)
print("AUC for SVC-radial : ",roc_auc_svc_radial)
```

```
AUC for SVC-radial :  0.8675450762829403
```

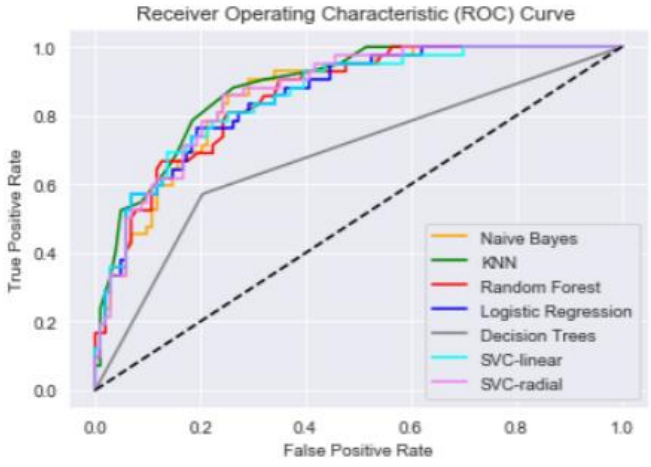
In [102]:
```
def plot_roc_curve(fpsvcradial,tpsvcradial):
    plt.plot(fpsvcradial,tpsvcradial,color='violet',label='ROC')
    plt.plot([0,1],[0,1],color='black',linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve ')
    plt.legend()
    plt.show
```

In [103]:
```
plot_roc_curve(fpsvcradial,tpsvcradial)
```




# MODEL COMPARISON

In [104]:
```
plt.plot(fpnb,tpnb,color='orange',label='Naive Bayes')
plt.plot(fpknn,tpknn,color='green',label='KNN')
plt.plot(fprf,tprf,color='red',label='Random Forest')
plt.plot(fplr,tplr,color='blue',label='Logistic Regression')
plt.plot(fpdt,tpdt,color='grey',label='Decision Trees')
plt.plot(fpsvclinear,tpsvclinear,color='cyan',label='SVC-linear')
plt.plot(fpsvcradial,tpsvcradial,color='violet',label='SVC-radial')
plt.plot([0,1],[0,1],color='black',linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve ')
plt.legend()
plt.show
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```



In [105]:
```
data={'Model':['Gaussian NB','K-Nearest Neighbours','Random Forest','Logistic','Decision Trees','SVC-linear','SVC-radial'],
     'Accuracy':[accugaussNB,accuKNN,accuRF,accuLR,accuDT,accuSVC_linear,accuSVC_radial],
     'AUC':[roc_auc_nb,roc_auc_knn,roc_auc_rf,roc_auc_lr,roc_auc_dt,roc_auc_svc_linear,roc_auc_svc_radial],
     'Sensitivity':[anb,aknn,arf,alr,adt,asvclinear,asvcradial],'Specificity':[bnb,bknn,brf,blr,bdt,bsvclinear,bsvcradial]}
df=pd.DataFrame(data)
df_new=df.style.set_properties(**{'text-align':'left'})
df_new
```

Out[105]:

| | Model | Accuracy | AUC | Sensitivity | Specificity |
|---|---|---|---|---|---|
| 0 | Gaussian NB | 0.793103 | 0.860610 | 0.873786 | 0.595238 |
| 1 | K-Nearest Neighbours | 0.827586 | 0.882917 | 0.951456 | 0.523810 |
| 2 | Random Forest | 0.793103 | 0.860379 | 0.885714 | 0.571429 |
| 3 | Logistic | 0.813793 | 0.856681 | 0.912621 | 0.571429 |
| 4 | Decision Trees | 0.696552 | 0.645400 | 0.786408 | 0.571429 |
| 5 | SVC-linear | 0.813793 | 0.858067 | 0.912621 | 0.571429 |
| 6 | SVC-radial | 0.806897 | 0.867545 | 0.932039 | 0.500000 |

# CONCLUSION

The objective of this project was to predict diabetes using 7 basic statistical learning techniques. We can see from the above comparison that for our data, K-NN (K- Nearest Neighbours) is the best classifier for the data. It's Accuracy, AUC, and Sensitivity, all rank better than the other models. It can accurately classify the patients 82.76% of times. The combined ROC graph helps us to visualize this comparison. The combined ROC graph helps us to visualize this comparison. Out of the seven models, decision trees can be considered less efficient. Note that K-NN's ROC has fewer steps that the others because its TPR and FPR was calculated only for 30 values of K.

# SCOPE OF THE PROJECT

This project can help to predict the accuracy of different statistical models and comparative study of them in order to predict the predisposition of a subject to diabetes. Such is the power of statistical learning that without much medical knowledge, and relying just on past patients' data, a statistician is in a position to build surprisingly accurate prediction models. A possible application would be to create an app that predicts diabetes. This would help to reach out to masses and to spread awareness. Methods used in this project can be applied to other areas of medicine as well. The scope of the project goes as far as one can take it with better understanding of machine learning, improved techniques and sophisticated algorithms, thus transforming the future of medical diagnosis for the better.

# REFERENCES

This project would have been impossible without Josh Starmer from Stat Quest and Krish Naik on YouTube. Their tutorials on machine learning techniques helped clear our basics, enabling us to take up this project.


- www.cdc.gov/diabetes/basics/gestational.html
- https://youtube.be/HVXime0nQel
- https://youtube/IDCWX6vCLFA
- www.kaggle.com/uciml/pima-indians-diabetes-database
- https://en.m.wikipedia.org/wiki/Naive-Bayes-classifier
- https://www.analyticsvidhya.com/blog2017/09/naive-bayes-explained/
- https://www.geeksforgeeks.org/naive-bayes-classifier
- https://medium.com/@srishtisawla/introduction-to-naive-bayes-for-classification-baefefb43a2d
- https://towardsdatascience.com/pima-indian-diabetes-prediction-7573698bd5fe
- https://www.biorxiv.org/content/10.1101/643833v2.full
- https://www.analyticsvidhya.com/blog/202006/auc-roc-curve-machine-learning
- https://youtu.be/XSoau-q0kz8
- https://youtu.be/lDCWX6vCLFA
- SVM | Support Vector Machine Algorithm in Machine Learning (analyticsvidhya.com)
- https://www.youtube.com/watch?v=EuBBz3bI-aA
- https://www.youtube.com/watch?v=efR1C6CvhmE&t=895s
- https://www.youtube.com/watch?v=Toet3EiSFcM
- https://www.youtube.com/watch?v=Qc5IyLW_hns