Healthcare DESCRIPTION ** Problem Statement *** NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases. *** The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has *** diabetes, based on certain diagnostic measurements included in the dataset. *** Build a model to accurately predict whether the patients in the dataset have diabetes or not. ** Dataset Description ***The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more. // Variables Description /// Pregnancies : Number of times pregnant /// Glucose Plasma : glucose concentration in an oral glucose tolerance test /// BloodPressure : Diastolic blood pressure (mm Hg) /// SkinThickness : Triceps skinfold thickness (mm) /// Insulin : Two hour serum insulin /// BMI : Body Mass Index ///DiabetesPedigreeFunction: Diabetes pedigree function /// Age : Age in years /// Outcome : Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0 * Project Task: Week 1 * ** Data Exploration: 1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value: • Glucose • BloodPressure • SkinThickness • Insulin • BMI 2. Visually explore these variables using histograms. Treat the missing values accordingly. 3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables. * Project Task: Week 2 * ** Data Exploration: ** 1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action. 2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings. 3. Perform correlation analysis. Visually explore it using a heat map. * Project Task: Week 3 * ** Data Modeling: ** 1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process. 2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm. * Project Task: Week 4 * ** Data Modeling: ** 1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used. // Data Reporting: 2. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following: a. Pie chart to describe the diabetic or non-diabetic population b. Scatter charts between relevant variables to analyze the relationships c. Histogram or frequency charts to analyze the distribution of the data d. Heatmap of correlation analysis among the relevant variables e. Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.

In [1]:
```
# importing the required libraries

import numpy as np
from math import sqrt

import pandas as pd
import pandas_profiling
from pandas_profiling import ProfileReport


import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

from sklearn.metrics import r2_score,accuracy_score,classification_report,confusion_
from sklearn.model_selection import train_test_split
```

In [2]:
```
# importing the diabetics/Non-diabetics dataset

data = pd.read_csv("health care diabetes.csv")
```

## Data Exploration:

### 1.Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

Glucose

BloodPressure

SkinThickness

Insulin

BMI

In [3]:

```
# first 5 rows
data.head()
```

Out[3]:

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

In [4]:

```
data.shape
```

Out[4]: (768, 9)

In [5]:

```
data.info()
```

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

In [6]:

```
data.describe()
```

Out[6]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedig |
|---|---|---|---|---|---|---|---|
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | |
| **mean** | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | |
| **std** | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | |

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedig |
|---|---|---|---|---|---|---|---|
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | |

In [7]:
```
data.isnull().any()
```

Out[7]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

## Our dataset contains 768 rows , 9 columns , Outcome = Target/result , 2 columns are Float data type and 7 are int data type

In [8]:
```
profile_report_data = ProfileReport(data)
profile_report_data
```

Out[8]:

In [9]:

```
for i in data.columns:
    print("mean,median of {} column is : {} , {}".format(i,data[i].mean(),data[i].me
```

```
mean,median of Pregnancies column is : 3.8450520833333335 , 3.0
mean,median of Glucose column is : 120.89453125 , 117.0
mean,median of BloodPressure column is : 69.10546875 , 72.0
mean,median of SkinThickness column is : 20.536458333333332 , 23.0
mean,median of Insulin column is : 79.79947916666667 , 30.5
mean,median of BMI column is : 31.992578124999977 , 32.0
mean,median of DiabetesPedigreeFunction column is : 0.4718763020833327 , 0.3725
```

```
mean,median of Age column is : 33.240885416666664 , 29.0
mean,median of Outcome column is : 0.3489583333333333 , 0.0
```

In [10]:

```
for i in data.columns:
    print("value counts of {} is:{}".format(i,data[i].value_counts()))
```

```
value counts of Pregnancies is:1     135
0     111
2     103
3      75
4      68
5      57
6      50
7      45
8      38
9      28
10     24
11     11
13     10
12      9
14      2
15      1
17      1
Name: Pregnancies, dtype: int64
value counts of Glucose is:99     17
100    17
129    14
125    14
106    14
       ..
169     1
61      1
178     1
177     1
199     1
Name: Glucose, Length: 136, dtype: int64
value counts of BloodPressure is:70     57
74     52
78     45
68     45
72     44
64     43
80     40
76     39
60     37
0      35
62     34
82     30
66     30
88     25
84     23
90     22
58     21
86     21
50     13
56     12
54     11
52     11
92      8
75      8
65      7
85      6
94      6
48      5
96      4
44      4
110     3
106     3
```

```
100     3
98      3
108     2
104     2
46      2
55      2
30      2
95      1
61      1
102     1
38      1
40      1
24      1
114     1
122     1
Name: BloodPressure, dtype: int64
value counts of SkinThickness is:0     227
32     31
30     27
27     23
23     22
18     20
28     20
33     20
31     19
19     18
39     18
29     17
40     16
37     16
22     16
25     16
26     16
41     15
35     15
36     14
15     14
17     14
20     13
24     12
13     11
42     11
21     10
46      8
34      8
12      7
38      7
11      6
45      6
16      6
14      6
43      6
44      5
10      5
47      4
48      4
49      3
50      3
54      2
52      2
8       2
7       2
51      1
56      1
60      1
63      1
99      1
Name: SkinThickness, dtype: int64
value counts of Insulin is:0      374
```

```
105     11
140      9
130      9
120      8
        ...
193      1
191      1
188      1
184      1
846      1
Name: Insulin, Length: 186, dtype: int64
value counts of BMI is:32.0    13
31.2    12
31.6    12
0.0     11
33.3    10
        ..
19.3     1
49.3     1
19.4     1
20.0     1
40.1     1
Name: BMI, Length: 248, dtype: int64
value counts of DiabetesPedigreeFunction is:0.258    6
0.254    6
0.268    5
0.261    5
0.207    5
        ..
0.145    1
0.241    1
1.292    1
0.627    1
0.804    1
Name: DiabetesPedigreeFunction, Length: 517, dtype: int64
value counts of Age is:22    72
21    63
25    48
24    46
23    38
28    35
26    33
27    32
29    29
31    24
41    22
30    21
37    19
42    18
33    17
38    16
36    16
32    16
45    15
34    14
40    13
43    13
46    13
39    12
35    10
52     8
44     8
50     8
51     8
58     7
54     6
47     6
53     5
60     5
```

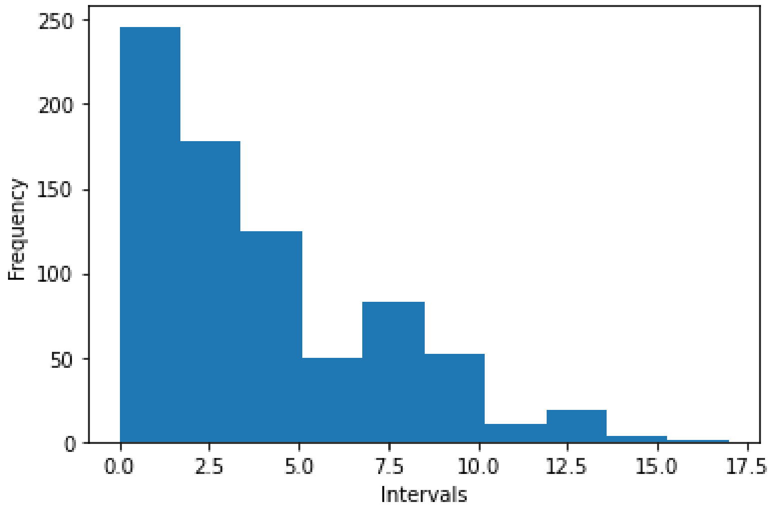
```
49      5
57      5
48      5
66      4
62      4
63      4
55      4
59      3
56      3
65      3
67      3
61      2
69      2
64      1
68      1
70      1
72      1
81      1
Name: Age, dtype: int64
value counts of Outcome is:0    500
1    268
Name: Outcome, dtype: int64
```

## Visually explore these variables using histograms

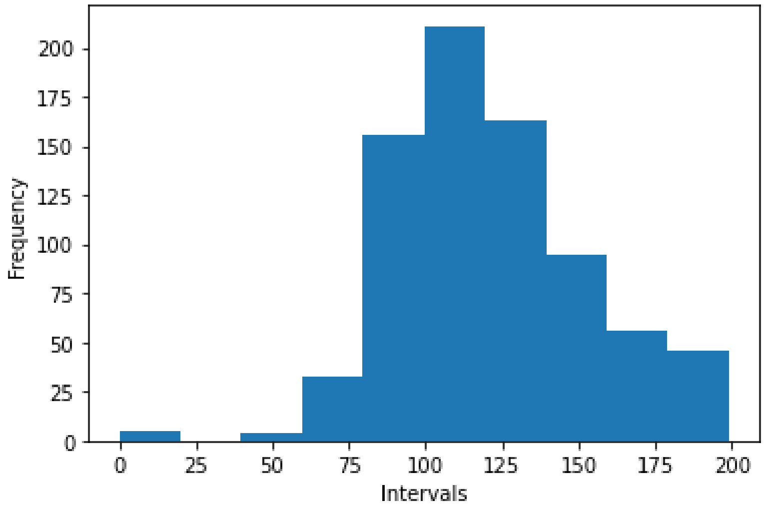
In [11]:

```
# in single step for viewing all histogram graphs

for i in data.columns:
    ##data[i].plot(kind='hist')
    print("Histogram of {} is {}".format(i,data[i].plot(kind='hist')))
    print("{}".format(plt.xlabel("Intervals")))
    plt.show()
```

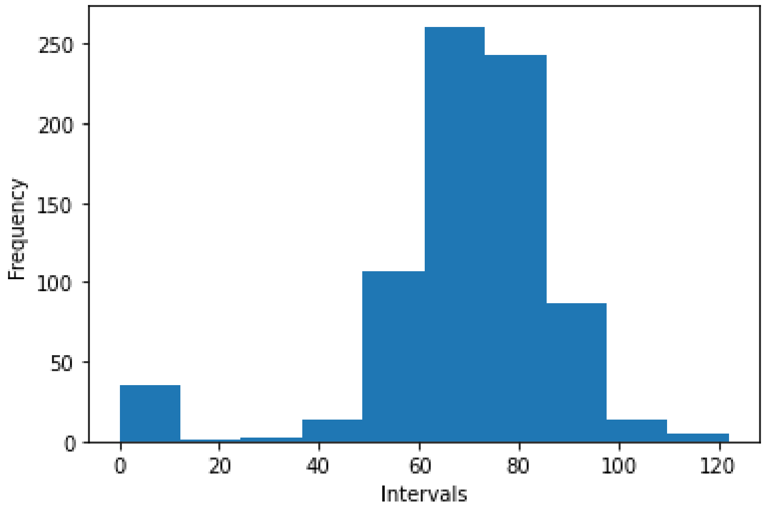
```
Histogram of Pregnancies is AxesSubplot(0.125,0.125;0.775x0.755)
Text(0.5, 0, 'Intervals')
```



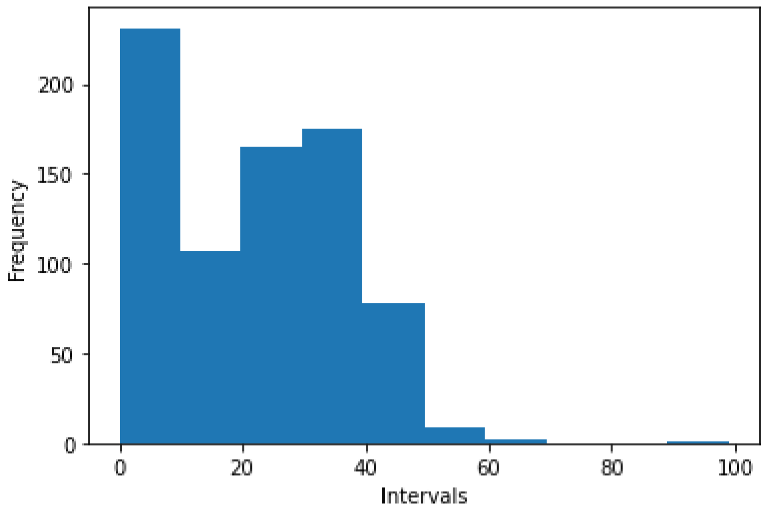

```
Histogram of Glucose is AxesSubplot(0.125,0.125;0.775x0.755)
Text(0.5, 0, 'Intervals')
```



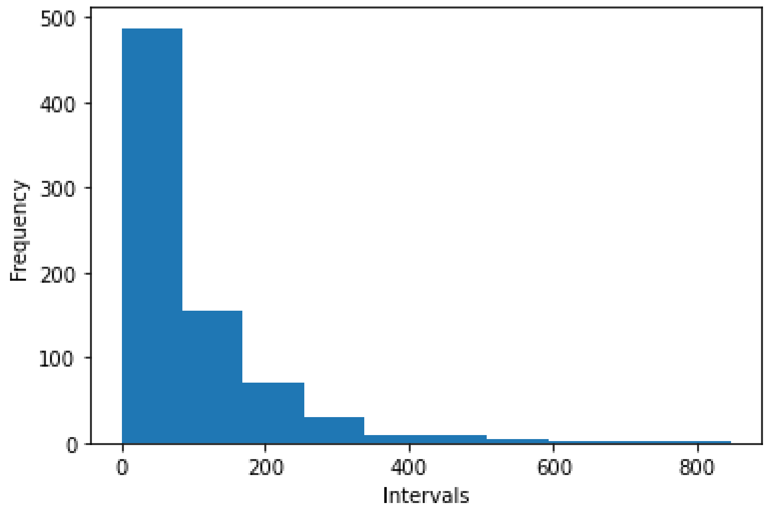

```
Histogram of BloodPressure is AxesSubplot(0.125,0.125;0.775x0.755)
Text(0.5, 0, 'Intervals')
```



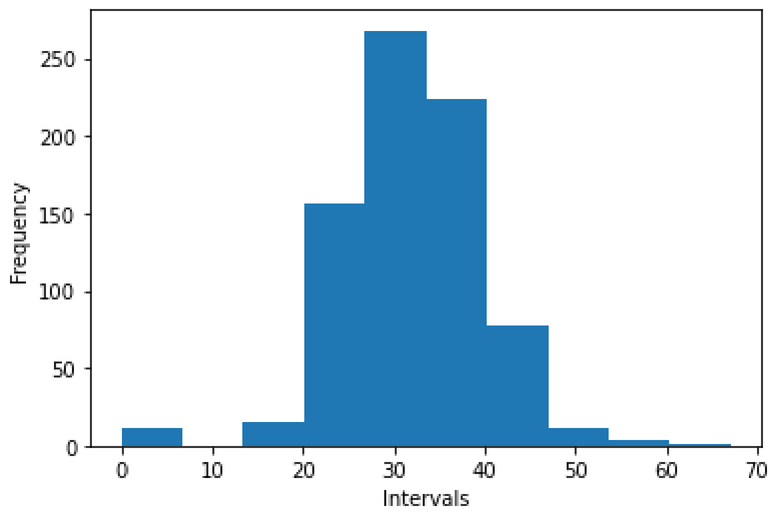

```
Histogram of SkinThickness is AxesSubplot(0.125,0.125;0.775x0.755)
Text(0.5, 0, 'Intervals')
```



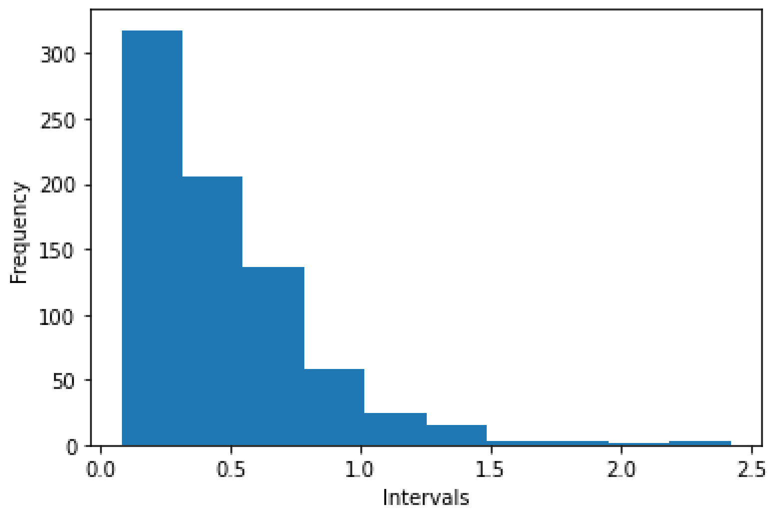

```
Histogram of Insulin is AxesSubplot(0.125,0.125;0.775x0.755)
Text(0.5, 0, 'Intervals')
```



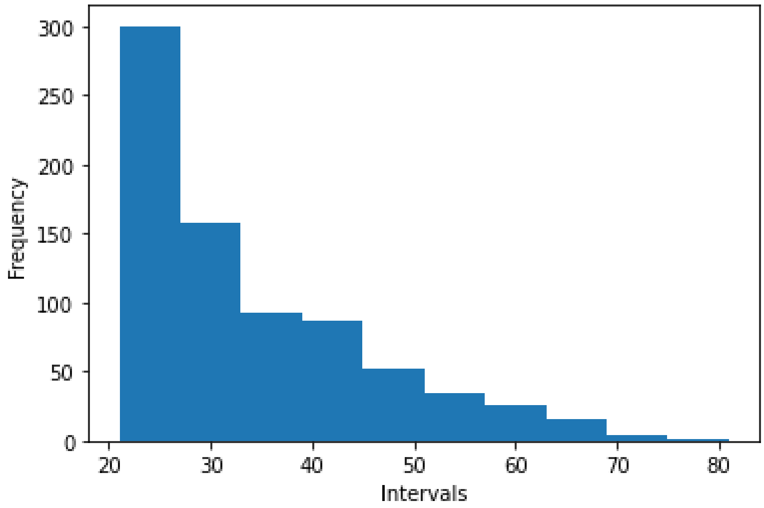

```
Histogram of BMI is AxesSubplot(0.125,0.125;0.775x0.755)
Text(0.5, 0, 'Intervals')
```



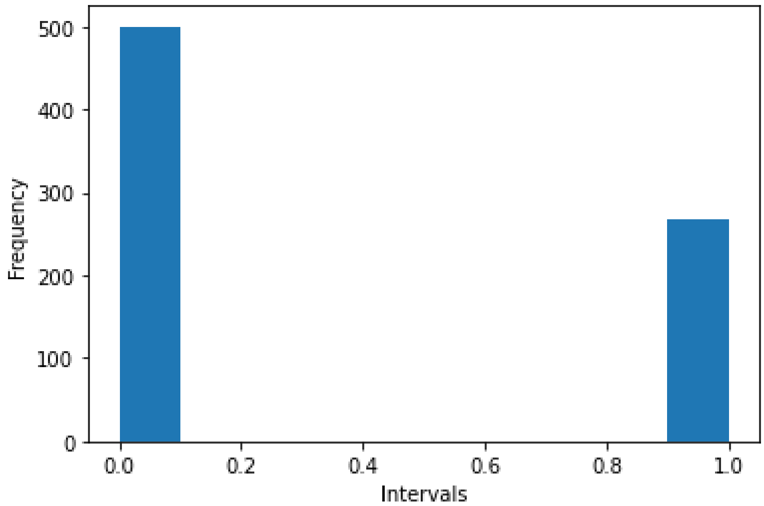

```
Histogram of DiabetesPedigreeFunction is AxesSubplot(0.125,0.125;0.775x0.755)
Text(0.5, 0, 'Intervals')
```




```
Histogram of Age is AxesSubplot(0.125,0.125;0.775x0.755)
Text(0.5, 0, 'Intervals')
```




```
Histogram of Outcome is AxesSubplot(0.125,0.125;0.775x0.755)
Text(0.5, 0, 'Intervals')
```




## Treat the missing values accordingly

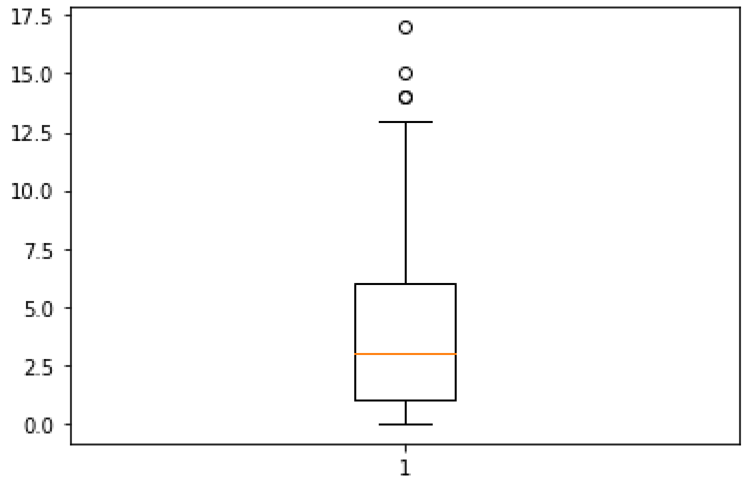
In [12]:

```
# as per the data On the columns below, a value of zero does not make sense and thus

### Glucose,BloodPressure,SkinThickness,Insulin,BMI
```

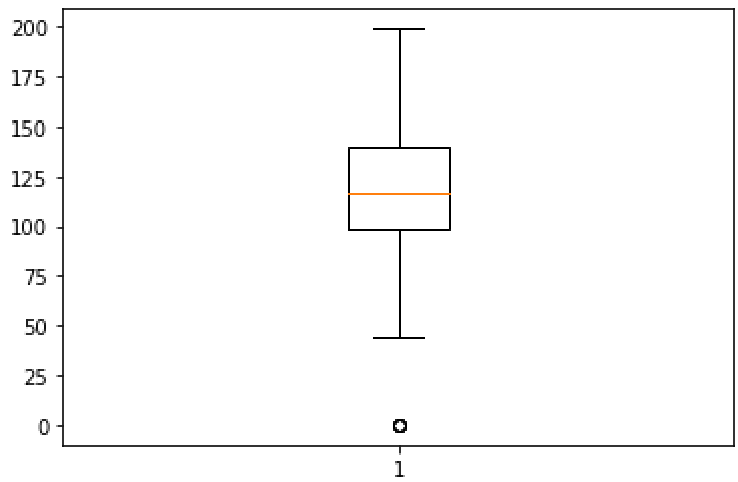
In [13]:

```
for i in data.columns:
    print(" Boxplot of {}:{}".format(i,plt.boxplot(data[i])))
    plt.show()
```

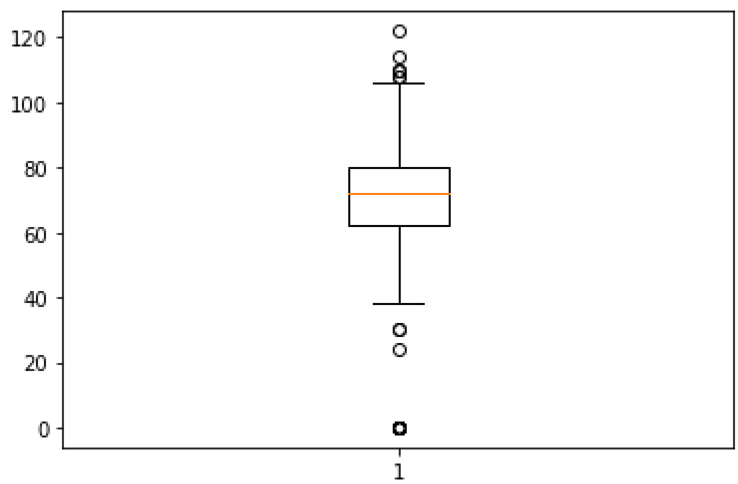
```
 Boxplot of Pregnancies:{'whiskers': [<matplotlib.lines.Line2D object at 0x000001F81D4E79A0>, <matplotlib.lines.Line2D object at 0x000001F81D4E7D00>], 'caps': [<matplotlib.lines.Line2D object at 0x000001F81D4F60A0>, <matplotlib.lines.Line2D object at 0x000001F81D4F6400>], 'boxes': [<matplotlib.lines.Line2D object at 0x000001F81D4E7640>], 'medians': [<matplotlib.lines.Line2D object at 0x000001F81D4F6760>], 'fliers': [<matplotlib.lines.Line2D object at 0x000001F81D4F6AC0>], 'means': []}
```



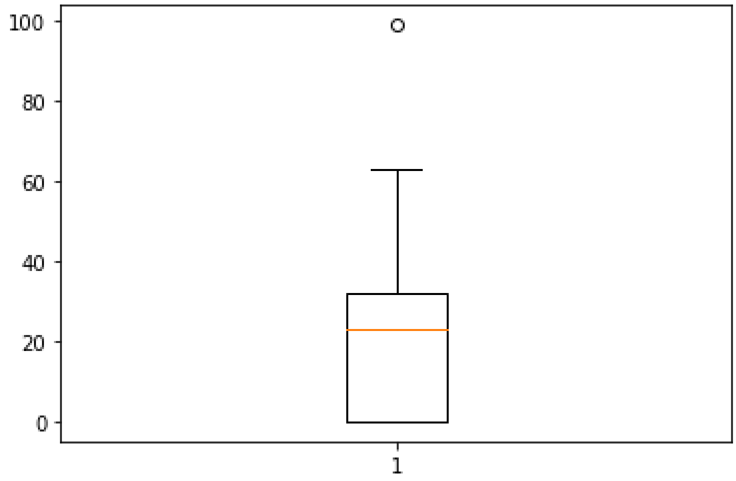

```
 Boxplot of Glucose:{'whiskers': [<matplotlib.lines.Line2D object at 0x000001F81D546
BB0>, <matplotlib.lines.Line2D object at 0x000001F81D546F10>], 'caps': [<matplotlib.
lines.Line2D object at 0x000001F81D5552B0>, <matplotlib.lines.Line2D object at 0x000
001F81D555610>], 'boxes': [<matplotlib.lines.Line2D object at 0x000001F81D546850>],
'medians': [<matplotlib.lines.Line2D object at 0x000001F81D555970>], 'fliers': [<mat
plotlib.lines.Line2D object at 0x000001F81D555CD0>], 'means': []}
```



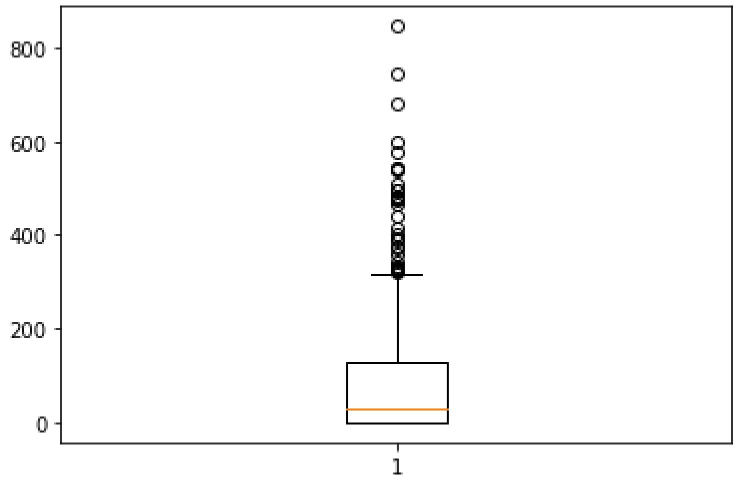

```
 Boxplot of BloodPressure:{'whiskers': [<matplotlib.lines.Line2D object at 0x000001F
81D5AF2B0>, <matplotlib.lines.Line2D object at 0x000001F81D5AF610>], 'caps': [<matpl
otlib.lines.Line2D object at 0x000001F81D5AF970>, <matplotlib.lines.Line2D object at
0x000001F81D5AFCD0>], 'boxes': [<matplotlib.lines.Line2D object at 0x000001F81D5A0F1
0>], 'medians': [<matplotlib.lines.Line2D object at 0x000001F81D5BD070>], 'fliers':
[<matplotlib.lines.Line2D object at 0x000001F81D5BD3D0>], 'means': []}
```



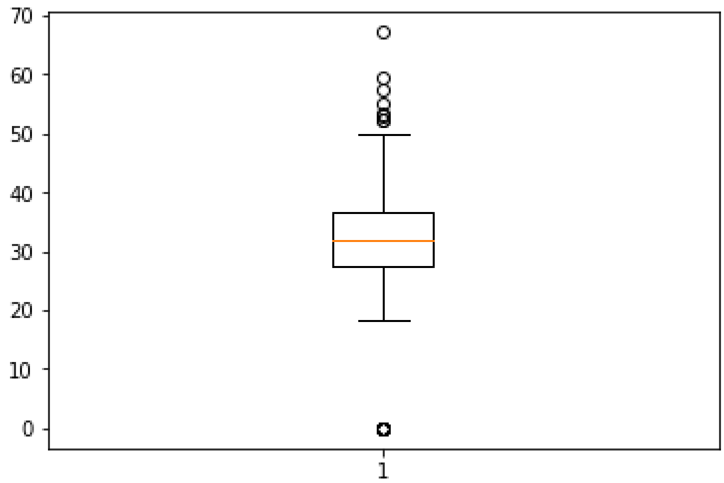

```
 Boxplot of SkinThickness:{'whiskers': [<matplotlib.lines.Line2D object at 0x000001F
81D600FD0>, <matplotlib.lines.Line2D object at 0x000001F81D60F370>], 'caps': [<matpl
otlib.lines.Line2D object at 0x000001F81D60F6D0>, <matplotlib.lines.Line2D object at
0x000001F81D60FA30>], 'boxes': [<matplotlib.lines.Line2D object at 0x000001F81D600C7
0>], 'medians': [<matplotlib.lines.Line2D object at 0x000001F81D60FD90>], 'fliers':
[<matplotlib.lines.Line2D object at 0x000001F81D61B130>], 'means': []}
```



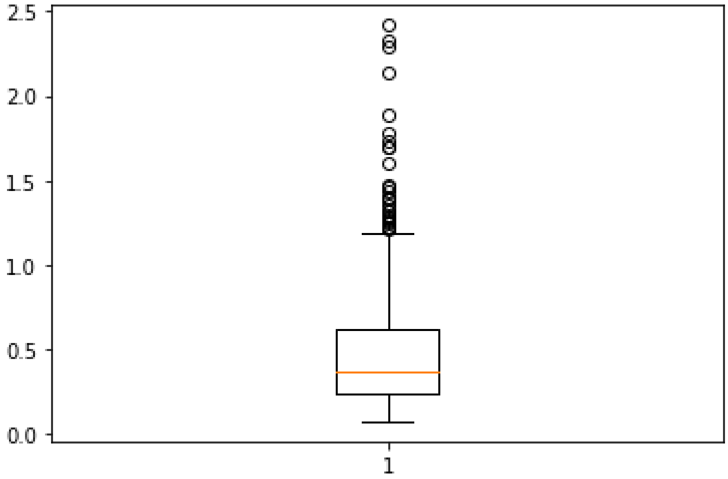

```
 Boxplot of Insulin:{'whiskers': [<matplotlib.lines.Line2D object at 0x000001F81D661
880>, <matplotlib.lines.Line2D object at 0x000001F81D661BE0>], 'caps': [<matplotlib.
lines.Line2D object at 0x000001F81D661F40>, <matplotlib.lines.Line2D object at 0x000
001F81D6702E0>], 'boxes': [<matplotlib.lines.Line2D object at 0x000001F81D661520>],
'medians': [<matplotlib.lines.Line2D object at 0x000001F81D670640>], 'fliers': [<mat
plotlib.lines.Line2D object at 0x000001F81D6709A0>], 'means': []}
```



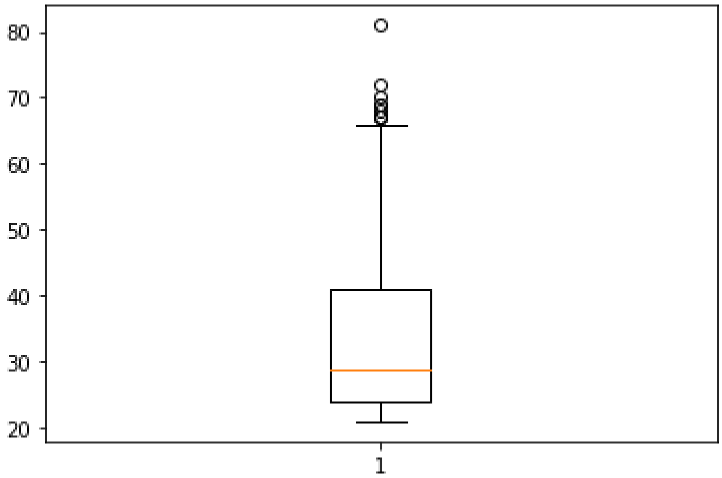

```
 Boxplot of BMI:{'whiskers': [<matplotlib.lines.Line2D object at 0x000001F81D6B6D60
>, <matplotlib.lines.Line2D object at 0x000001F81D6C60D0>], 'caps': [<matplotlib.lin
es.Line2D object at 0x000001F81D6C6400>, <matplotlib.lines.Line2D object at 0x000001
F81D6C6730>], 'boxes': [<matplotlib.lines.Line2D object at 0x000001F81D6B6B20>], 'me
dians': [<matplotlib.lines.Line2D object at 0x000001F81D6C6A60>], 'fliers': [<matplo
tlib.lines.Line2D object at 0x000001F81D6C6D90>], 'means': []}
```



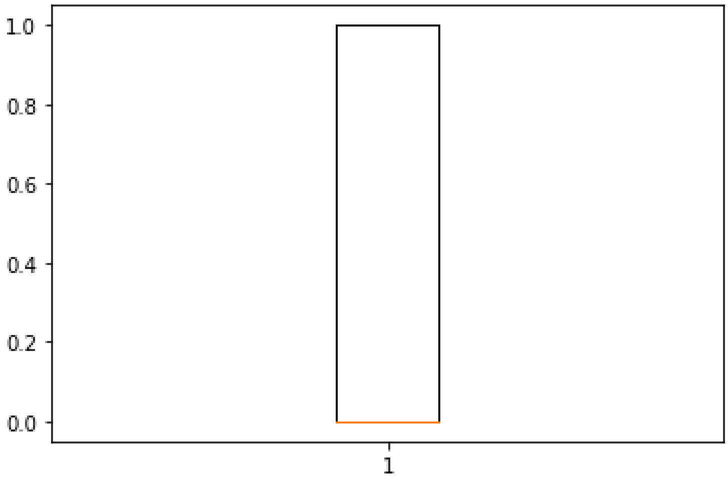

```
 Boxplot of DiabetesPedigreeFunction:{'whiskers': [<matplotlib.lines.Line2D object a
t 0x000001F81D715E50>, <matplotlib.lines.Line2D object at 0x000001F81D7251F0>], 'cap
s': [<matplotlib.lines.Line2D object at 0x000001F81D725550>, <matplotlib.lines.Line2
D object at 0x000001F81D7258B0>], 'boxes': [<matplotlib.lines.Line2D object at 0x000
001F81D715AF0>], 'medians': [<matplotlib.lines.Line2D object at 0x000001F81D725C10
>], 'fliers': [<matplotlib.lines.Line2D object at 0x000001F81D725F70>], 'means': []}
```




```
Boxplot of Age:{'whiskers': [<matplotlib.lines.Line2D object at 0x000001F81D77A700>, <matplotlib.lines.Line2D object at 0x000001F81D77AA60>], 'caps': [<matplotlib.lines.Line2D object at 0x000001F81D77ADC0>, <matplotlib.lines.Line2D object at 0x000001F81D786160>], 'boxes': [<matplotlib.lines.Line2D object at 0x000001F81D77A3A0>], 'medians': [<matplotlib.lines.Line2D object at 0x000001F81D7864C0>], 'fliers': [<matplotlib.lines.Line2D object at 0x000001F81D786820>], 'means': []}
```




```
Boxplot of Outcome:{'whiskers': [<matplotlib.lines.Line2D object at 0x000001F81D7D7460>, <matplotlib.lines.Line2D object at 0x000001F81D7D77C0>], 'caps': [<matplotlib.lines.Line2D object at 0x000001F81D7D7B20>, <matplotlib.lines.Line2D object at 0x000001F81D7D7E80>], 'boxes': [<matplotlib.lines.Line2D object at 0x000001F81D7D7100>], 'medians': [<matplotlib.lines.Line2D object at 0x000001F81D7E4220>], 'fliers': [<matplotlib.lines.Line2D object at 0x000001F81D7E4580>], 'means': []}
```




In [14]:

```
# I can see there are some outliers, but I am not going to drop/remove them because
# Hence , I am just replacing the 0 with mean of the below mention columns
# Glucose,BloodPressure,SkinThickness,Insulin,BMI

data['Glucose'] = data.Glucose.replace(0,data['Glucose'].mean())
```

```
data['BloodPressure'] = data.BloodPressure.replace(0,data['BloodPressure'].mean())
data['SkinThickness'] = data.SkinThickness.replace(0,data['SkinThickness'].mean())
data['Insulin'] = data.Insulin.replace(0,data['Insulin'].mean())
data['BMI'] = data.BMI.replace(0,data['BMI'].mean())
```

In [15]:

```
# checking again if any 0's present in the Glucose,BloodPressure,SkinThickness,Insul

data[data['Glucose']==0]
```

Out[15]:

| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |
|---|---|---|---|---|---|---|---|

In [16]:

```
data[data['BloodPressure']==0]
```

Out[16]:

| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |
|---|---|---|---|---|---|---|---|

In [17]:

```
data[data['SkinThickness']==0]
```

Out[17]:

| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |
|---|---|---|---|---|---|---|---|

In [18]:

```
data[data['Insulin']==0]
```

Out[18]:

| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |
|---|---|---|---|---|---|---|---|

In [19]:

```
data[data['BMI']==0]
```

Out[19]:

| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |
|---|---|---|---|---|---|---|---|

Now, there are no missing values in our dataset / a value of zero does not make sense and thus indicates missing value. We are good

In [20]:

```
data.head()
```

Out[20]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72.0 | 35.000000 | 79.799479 | 33.6 | 0.627 |
| **1** | 1 | 85.0 | 66.0 | 29.000000 | 79.799479 | 26.6 | 0.351 |
| **2** | 8 | 183.0 | 64.0 | 20.536458 | 79.799479 | 23.3 | 0.672 |
| **3** | 1 | 89.0 | 66.0 | 23.000000 | 94.000000 | 28.1 | 0.167 |
| **4** | 0 | 137.0 | 40.0 | 35.000000 | 168.000000 | 43.1 | 2.288 |

# There are integer and float data type variables in this dataset.

## Create a count (frequency) plot describing the data types and the count of variables.

In [21]:
```
df  = pd.DataFrame(data)
```

In [22]:
```
df
```

Out[22]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunctio |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72.0 | 35.000000 | 79.799479 | 33.6 | 0.62 |
| **1** | 1 | 85.0 | 66.0 | 29.000000 | 79.799479 | 26.6 | 0.35 |
| **2** | 8 | 183.0 | 64.0 | 20.536458 | 79.799479 | 23.3 | 0.67 |
| **3** | 1 | 89.0 | 66.0 | 23.000000 | 94.000000 | 28.1 | 0.16 |
| **4** | 0 | 137.0 | 40.0 | 35.000000 | 168.000000 | 43.1 | 2.28 |
| **...** | ... | ... | ... | ... | ... | ... | . |
| **763** | 10 | 101.0 | 76.0 | 48.000000 | 180.000000 | 32.9 | 0.17 |
| **764** | 2 | 122.0 | 70.0 | 27.000000 | 79.799479 | 36.8 | 0.34 |
| **765** | 5 | 121.0 | 72.0 | 23.000000 | 112.000000 | 26.2 | 0.24 |
| **766** | 1 | 126.0 | 60.0 | 20.536458 | 79.799479 | 30.1 | 0.34 |
| **767** | 1 | 93.0 | 70.0 | 31.000000 | 79.799479 | 30.4 | 0.31 |

768 rows × 9 columns

In [23]:
```
df.dtypes
```

Out[23]:
```
Pregnancies                   int64
Glucose                     float64
BloodPressure               float64
SkinThickness               float64
Insulin                     float64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [24]:
```
a = df.select_dtypes('int64')
b= df.select_dtypes('float64')
```

In [25]:
```
int_coun = a.shape[1]
float_count = b.shape[1]
```

In [26]:
```
table = { 'datatypes':['Numeric(float)','integer'],
         'Count(Frequency)':[float_count,int_coun]
    }
```

In [27]:
```
table =  pd.DataFrame(table)
```

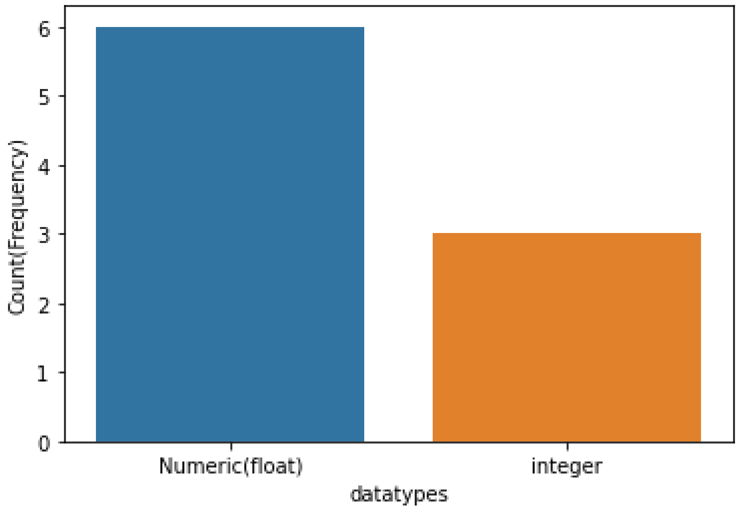
In [28]:
```
table
```

Out[28]:

| | datatypes | Count(Frequency) |
|---|---|---|
| **0** | Numeric(float) | 6 |
| **1** | integer | 3 |

In [29]:
```
sns.barplot(x= table.datatypes,y=table['Count(Frequency)'])
```

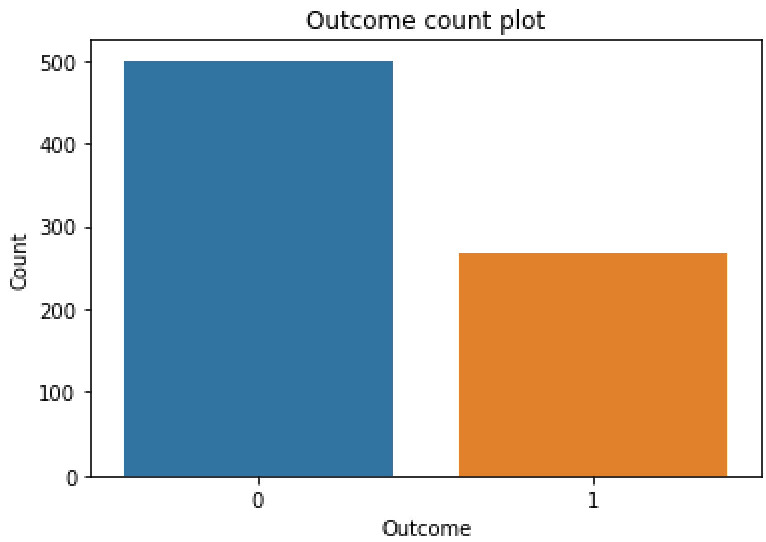
Out[29]:
```
<AxesSubplot:xlabel='datatypes', ylabel='Count(Frequency)'>
```




# Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

In [30]:
```
sns.countplot(data['Outcome'])
plt.title("Outcome count plot")
plt.xlabel("Outcome")
plt.ylabel("Count")
plt.show()
```

```
C:\Users\Pavan\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit k
eyword will result in an error or misinterpretation.
  warnings.warn(
```




In [31]:
```
data['Outcome'].value_counts()
```

Out[31]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

I can see that 500 are 0's means Non-diabetics , 268 are 1's means Diabetics

## Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

In [32]:
```
data
```

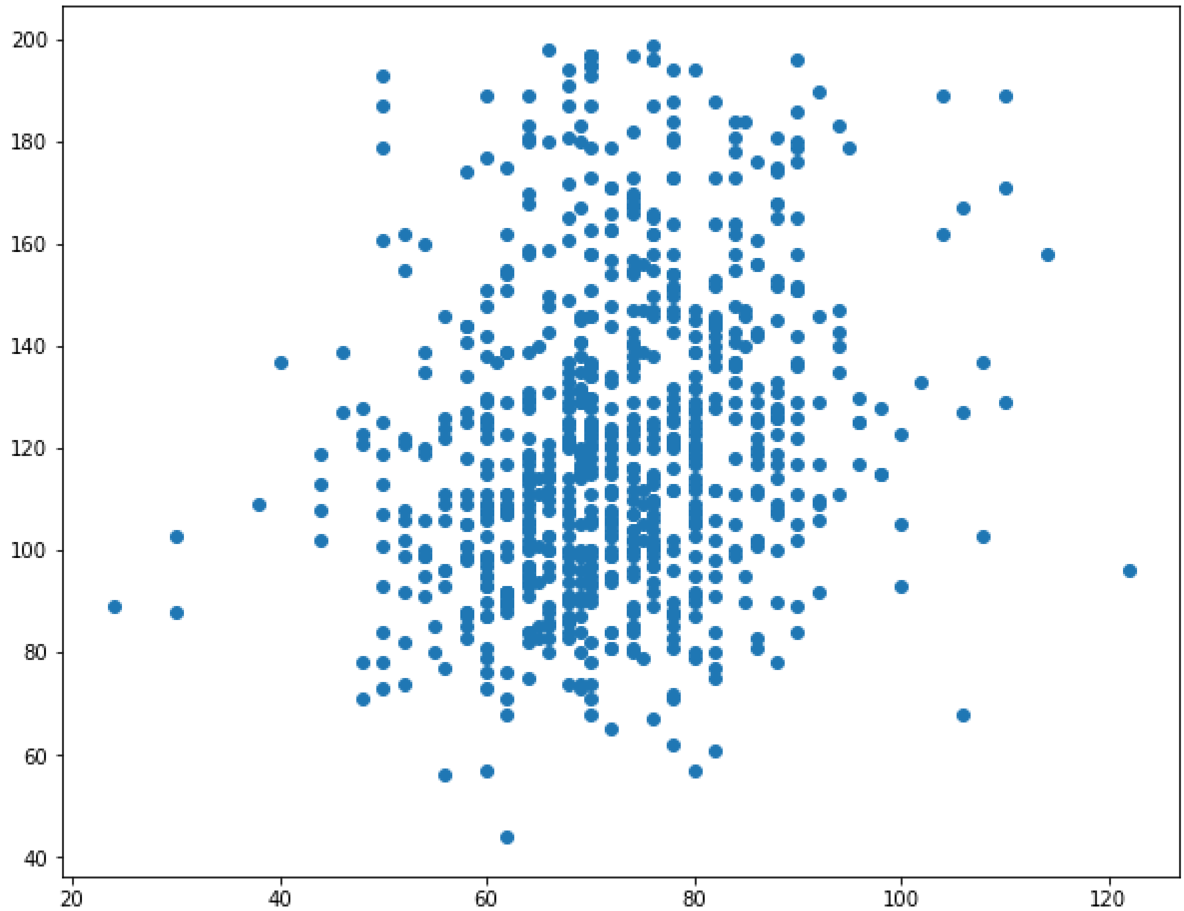
Out[32]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunctio |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72.0 | 35.000000 | 79.799479 | 33.6 | 0.62 |
| **1** | 1 | 85.0 | 66.0 | 29.000000 | 79.799479 | 26.6 | 0.35 |
| **2** | 8 | 183.0 | 64.0 | 20.536458 | 79.799479 | 23.3 | 0.67 |
| **3** | 1 | 89.0 | 66.0 | 23.000000 | 94.000000 | 28.1 | 0.16 |
| **4** | 0 | 137.0 | 40.0 | 35.000000 | 168.000000 | 43.1 | 2.28 |
| **...** | ... | ... | ... | ... | ... | ... | . |
| **763** | 10 | 101.0 | 76.0 | 48.000000 | 180.000000 | 32.9 | 0.17 |
| **764** | 2 | 122.0 | 70.0 | 27.000000 | 79.799479 | 36.8 | 0.34 |
| **765** | 5 | 121.0 | 72.0 | 23.000000 | 112.000000 | 26.2 | 0.24 |
| **766** | 1 | 126.0 | 60.0 | 20.536458 | 79.799479 | 30.1 | 0.34 |
| **767** | 1 | 93.0 | 70.0 | 31.000000 | 79.799479 | 30.4 | 0.31 |

768 rows × 9 columns

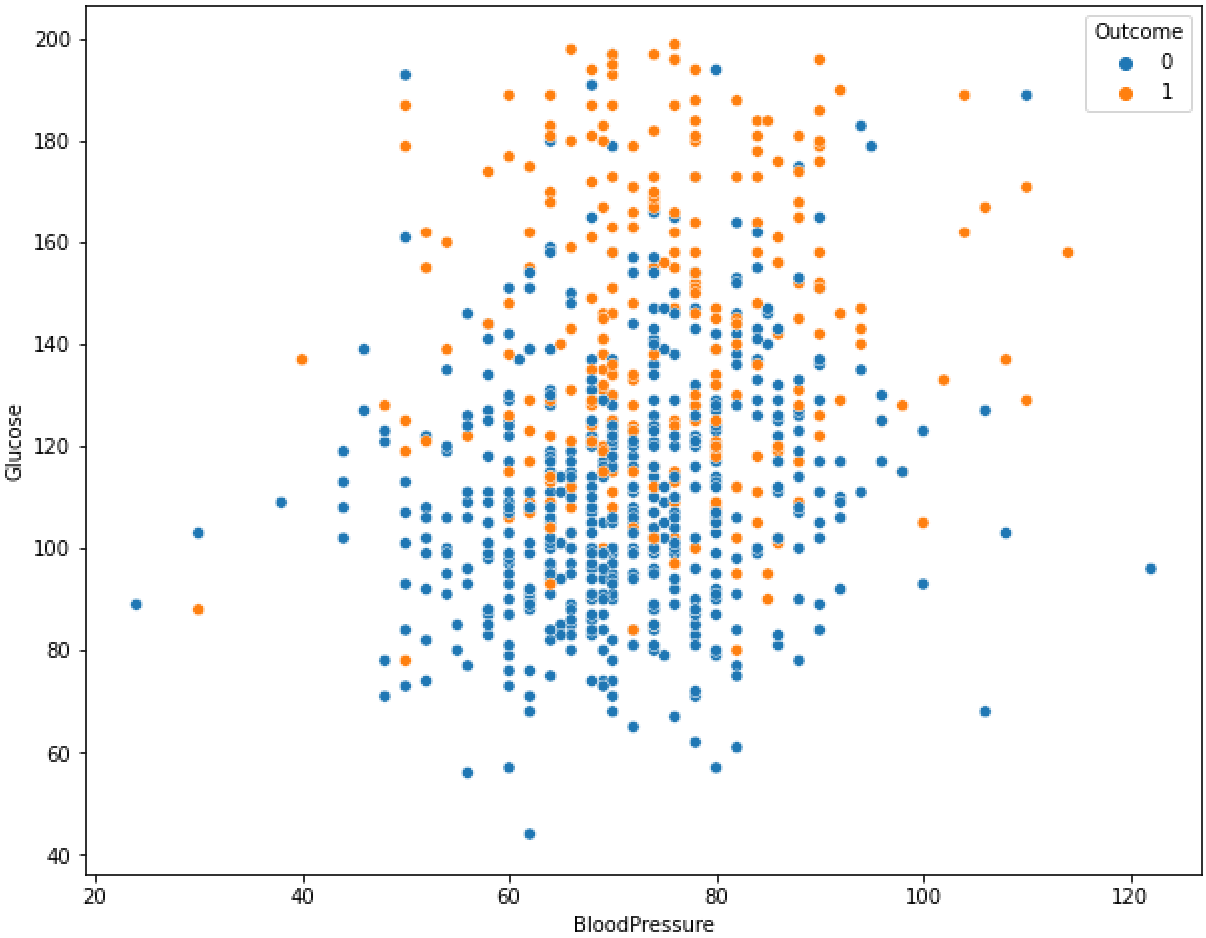
In [33]:
```
plt.figure(figsize=(10,8))
plt.scatter(x=data['BloodPressure'],y=data['Glucose'])
```

Out[33]:
```
<matplotlib.collections.PathCollection at 0x1f81d8dea90>
```




In [34]:
```
plt.figure(figsize=(10,8))
sns.scatterplot(x=data['BloodPressure'],y=data['Glucose'],hue=data['Outcome'])
```

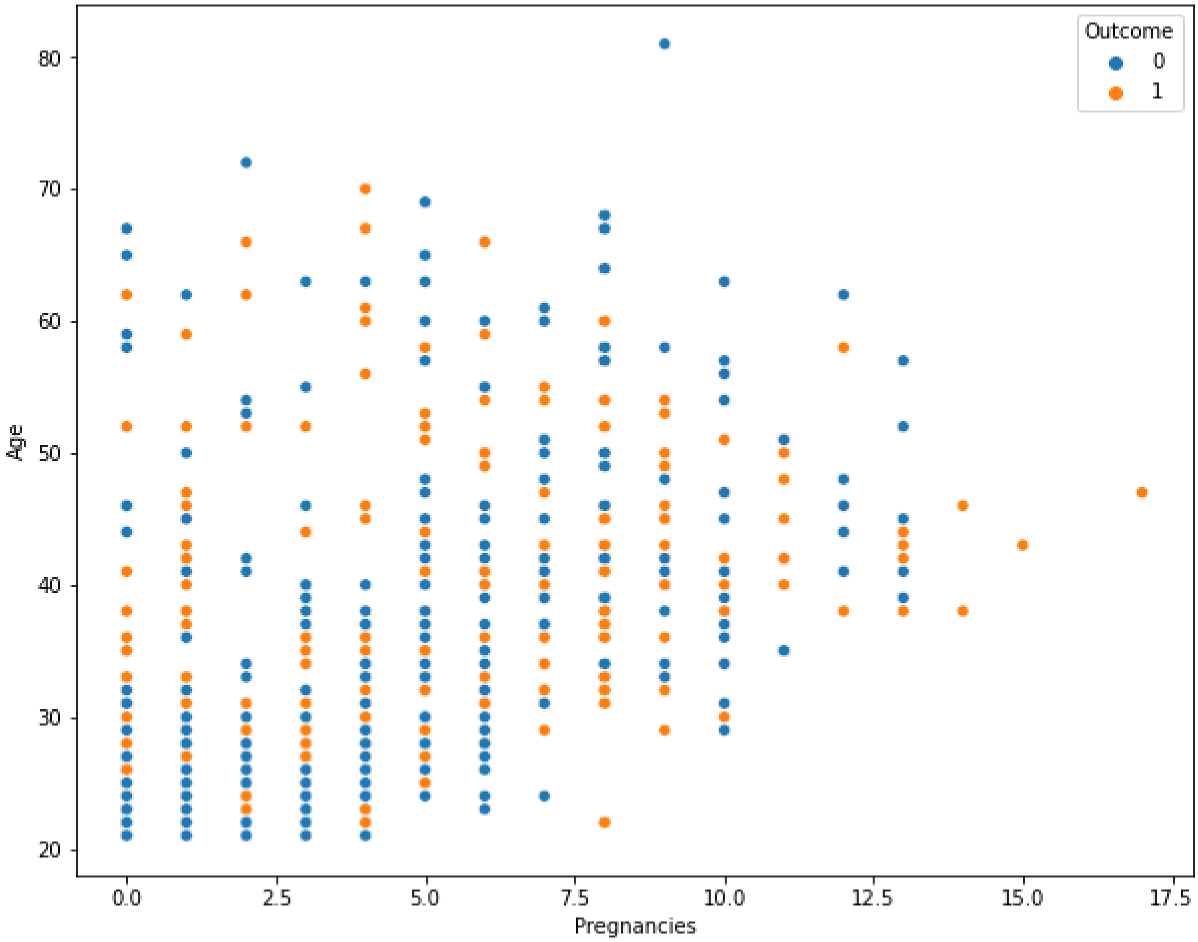
Out[34]: <AxesSubplot:xlabel='BloodPressure', ylabel='Glucose'>




In [35]:

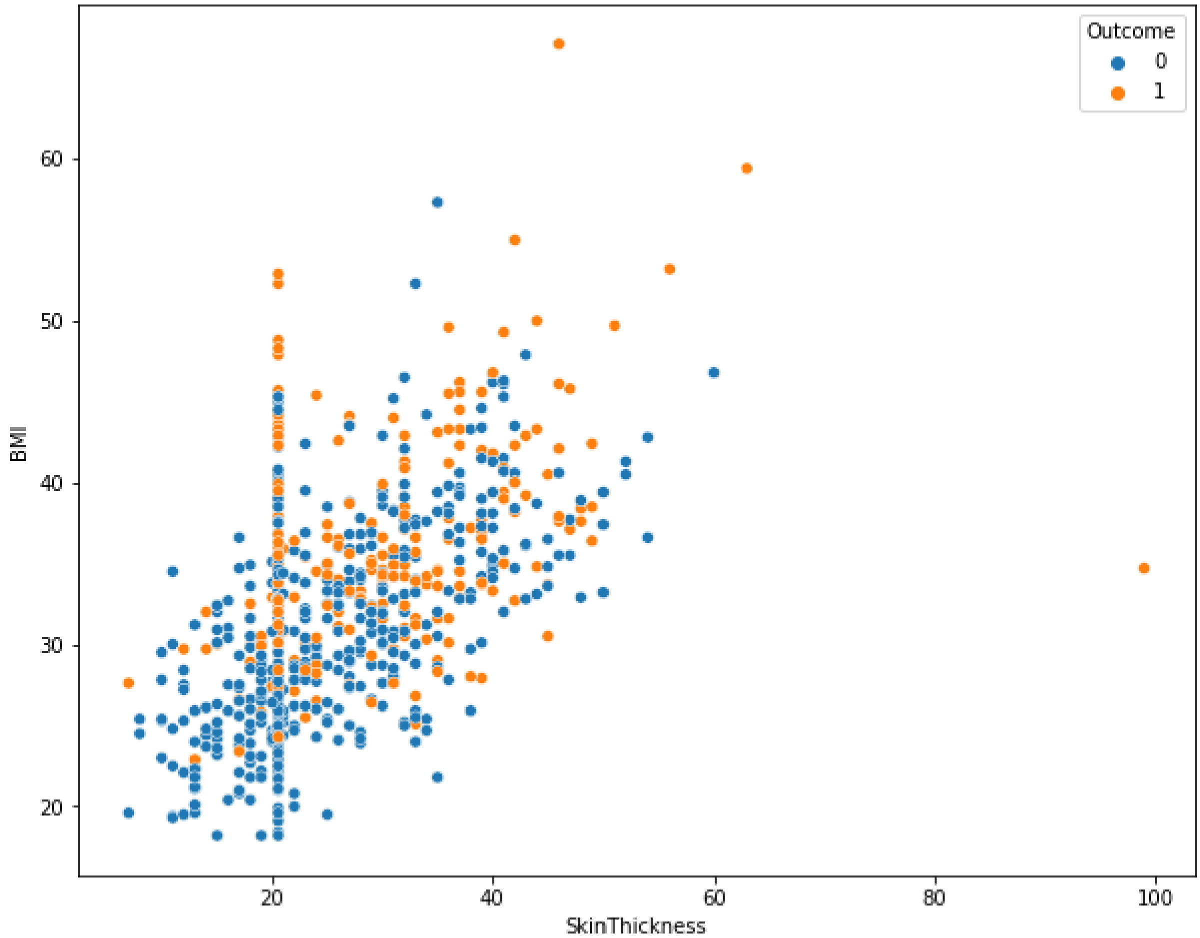
```
plt.figure(figsize=(10,8))
sns.scatterplot(x=data['Pregnancies'],y=data['Age'],hue=data['Outcome'])
```

Out[35]: <AxesSubplot:xlabel='Pregnancies', ylabel='Age'>



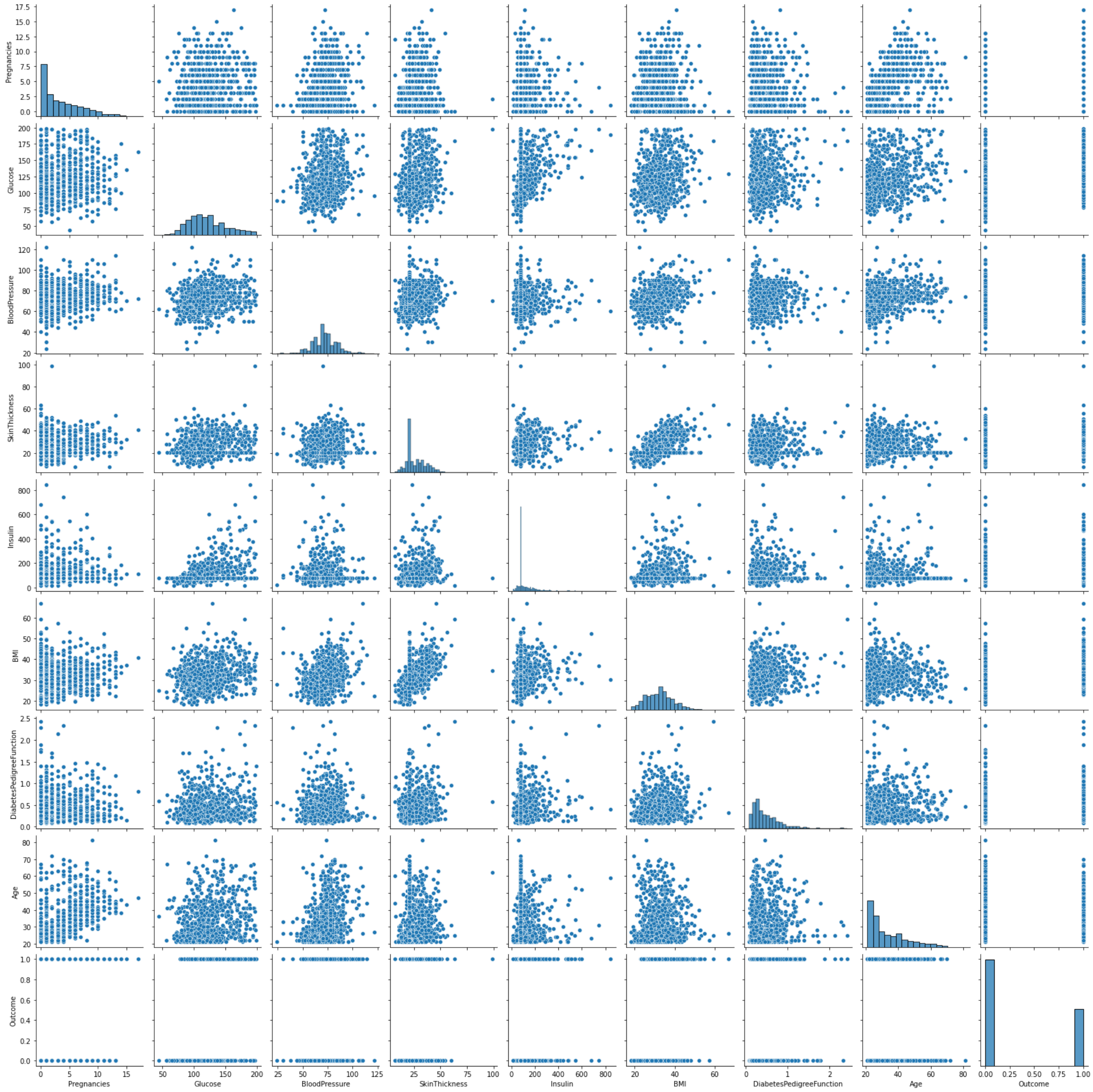

In [36]:
```
plt.figure(figsize=(10,8))
sns.scatterplot(x=data['SkinThickness'],y=data['BMI'],hue=data['Outcome'])
```

Out[36]: <AxesSubplot:xlabel='SkinThickness', ylabel='BMI'>




In [37]:
```
plt.figure(figsize=(20,18))
sns.pairplot(data)
```

Out[37]:
```
<seaborn.axisgrid.PairGrid at 0x1f81de7b9a0>
<Figure size 1440x1296 with 0 Axes>
```



I can see that there is some kind of +ve correlation in skinthickness - BMI , Pregnancy - age , Outcome has +ve correlation with Glucose ZC = zero correlation +ve = positive correlation -ve = negative correlation preg Gluc BP ST Insu BMI DPF Age O/c preg 1 ZC ZC ZC -ve ZC -ve +ve ZC Gluc ZC 1 ZC ZC ZC ZC ZC ZC +ve BP ZC ZC 1 ZC ZC ZC -ve ZC ZC ST ZC ZC ZC 1 -ve +ve ZC ZC ZC Insu ZC ZC -ve ZC 1 ZC ZC -ve ZC BMI ZC ZC ZC +ve ZC 1 ZC ZC ZC DPF ZC ZC -ve ZC ZC ZC 1 ZC ZC Age +ve ZC ZC ZC -ve ZC ZC 1 ZC O/c ZC +ve ZC ZC ZC ZC ZC ZC 1 I can see that there is no much multicollinearity. Let's explore

## Perform correlation analysis. Visually explore it using a heat map.

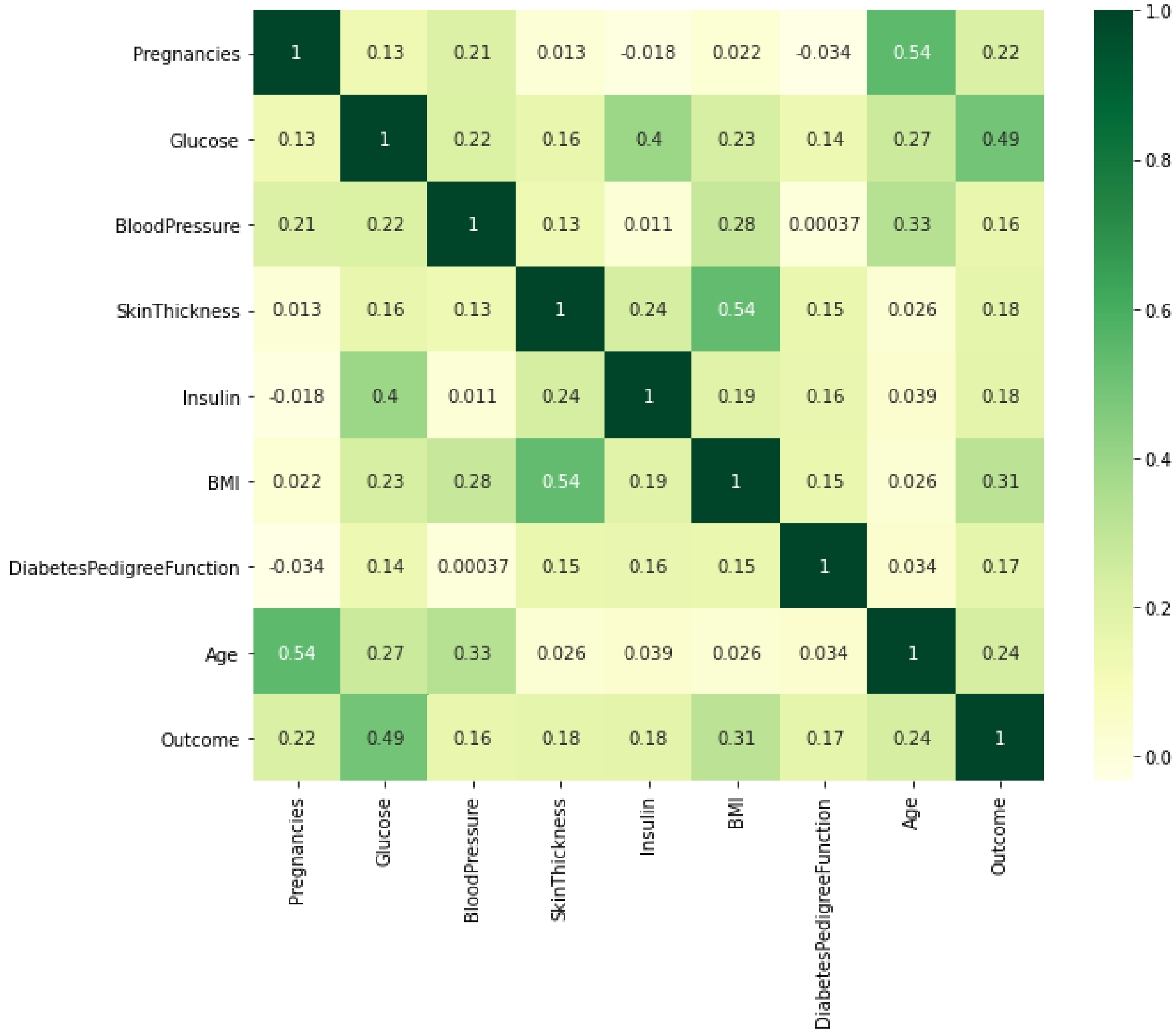
In [38]:
```
data.corr()
```

Out[38]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI |
|---|---|---|---|---|---|---|
| **Pregnancies** | 1.000000 | 0.127964 | 0.208984 | 0.013376 | -0.018082 | 0.021546 |
| **Glucose** | 0.127964 | 1.000000 | 0.219666 | 0.160766 | 0.396597 | 0.231478 |
| **BloodPressure** | 0.208984 | 0.219666 | 1.000000 | 0.134155 | 0.010926 | 0.281231 |
| **SkinThickness** | 0.013376 | 0.160766 | 0.134155 | 1.000000 | 0.240361 | 0.535703 |
| **Insulin** | -0.018082 | 0.396597 | 0.010926 | 0.240361 | 1.000000 | 0.189856 |
| **BMI** | 0.021546 | 0.231478 | 0.281231 | 0.535703 | 0.189856 | 1.000000 |

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI |
|---|---|---|---|---|---|---|
| **DiabetesPedigreeFunction** | -0.033523 | 0.137106 | 0.000371 | 0.154961 | 0.157806 | 0.153508 |
| **Age** | 0.544341 | 0.266600 | 0.326740 | 0.026423 | 0.038652 | 0.025748 |
| **Outcome** | 0.221898 | 0.492908 | 0.162986 | 0.175026 | 0.179185 | 0.312254 |

In [39]:
```
plt.figure(figsize=(10,8))
sns.heatmap(data.corr(),annot=True,cmap='YlGn')
plt.show()
```




# Project Task: Week 2

## Data Modeling:

1.Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

2.Apply an appropriate classification algorithm to build a model.

3.Compare various models with the results from KNN algorithm.

4.Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc.

## 5.Please be descriptive to explain what values of these parameter you have used.

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

Out[40]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72.0 | 35.000000 | 79.799479 | 33.6 | 0.627 |
| **1** | 1 | 85.0 | 66.0 | 29.000000 | 79.799479 | 26.6 | 0.351 |
| **2** | 8 | 183.0 | 64.0 | 20.536458 | 79.799479 | 23.3 | 0.672 |
| **3** | 1 | 89.0 | 66.0 | 23.000000 | 94.000000 | 28.1 | 0.167 |
| **4** | 0 | 137.0 | 40.0 | 35.000000 | 168.000000 | 43.1 | 2.288 |

Our dataset contains mostly Numerical , in such scenario Logistic Regression works fine ( other models also works well but Logistic can give better result)

By seeing our data our outcome variable has 2 values = 0/1 , Logistic classification algorithm will give best result.

I will be also using Support vector machine algorithm, KNN algorithm,Decision Tree classifier algorithm,Random Forest algorithm to see if i can improve the accuracy

Let's do this

In [275...
```
X = data.drop('Outcome',axis=1)
y = data['Outcome']
```

In [276...
```
X
```

Out[276...

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunctioi |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72.0 | 35.000000 | 79.799479 | 33.6 | 0.62 |
| **1** | 1 | 85.0 | 66.0 | 29.000000 | 79.799479 | 26.6 | 0.35 |
| **2** | 8 | 183.0 | 64.0 | 20.536458 | 79.799479 | 23.3 | 0.67 |
| **3** | 1 | 89.0 | 66.0 | 23.000000 | 94.000000 | 28.1 | 0.16 |
| **4** | 0 | 137.0 | 40.0 | 35.000000 | 168.000000 | 43.1 | 2.28 |
| **...** | ... | ... | ... | ... | ... | ... | . |
| **763** | 10 | 101.0 | 76.0 | 48.000000 | 180.000000 | 32.9 | 0.17 |
| **764** | 2 | 122.0 | 70.0 | 27.000000 | 79.799479 | 36.8 | 0.34 |
| **765** | 5 | 121.0 | 72.0 | 23.000000 | 112.000000 | 26.2 | 0.24 |
| **766** | 1 | 126.0 | 60.0 | 20.536458 | 79.799479 | 30.1 | 0.34 |
| **767** | 1 | 93.0 | 70.0 | 31.000000 | 79.799479 | 30.4 | 0.31 |

768 rows × 8 columns

In [277...
```
y
```

Out[277...
```
0      1
1      0
```

```
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [278…

```
x_train, x_test , y_train, y_test = train_test_split(X,y,test_size=0.25)
```

In [279…

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

```
(576, 8)
(192, 8)
(576,)
(192,)
```

# Logistic Regression

#### By analyzing the given dataset Outcome has 2 values i.e 0/1 . So I am using Logistic regression

In [280…

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
```

In [281…

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

```
C:\Users\Pavan\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Co
nvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

Out[281… LogisticRegression()

In [282…

```
log_reg.score(x_train,y_train)
```

Out[282… 0.7760416666666666

In [283…

```
log_reg.score(x_test,y_test)
```

Out[283… 0.765625

In [284…

```
# checking for the difference , difference = 0% , we are good now

log_reg.score(x_train,y_train) - log_reg.score(x_test,y_test)
```

Out[284… 0.01041666666666663

In [285…
```
prediction = log_reg.predict(x_test)
```

In [286…
```
log_reg.predict_proba(x_test)  # it will show probability of occurance
```

Out[286…
```
array([[0.31647716, 0.68352284],
       [0.86602558, 0.13397442],
       [0.92038217, 0.07961783],
       [0.71217564, 0.28782436],
       [0.71779712, 0.28220288],
       [0.88617305, 0.11382695],
       [0.55624966, 0.44375034],
       [0.49928958, 0.50071042],
       [0.90754282, 0.09245718],
       [0.75160464, 0.24839536],
       [0.33897715, 0.66102285],
       [0.60090243, 0.39909757],
       [0.71463009, 0.28536991],
       [0.92457022, 0.07542978],
       [0.66445674, 0.33554326],
       [0.23884476, 0.76115524],
       [0.10793719, 0.89206281],
       [0.89673044, 0.10326956],
       [0.15710231, 0.84289769],
       [0.95679523, 0.04320477],
       [0.93315506, 0.06684494],
       [0.92060286, 0.07939714],
       [0.35138992, 0.64861008],
       [0.75053719, 0.24946281],
       [0.73371507, 0.26628493],
       [0.57580725, 0.42419275],
       [0.80730434, 0.19269566],
       [0.95362003, 0.04637997],
       [0.59957938, 0.40042062],
       [0.87927494, 0.12072506],
       [0.80888809, 0.19111191],
       [0.90223603, 0.09776397],
       [0.63118451, 0.36881549],
       [0.81556848, 0.18443152],
       [0.94942624, 0.05057376],
       [0.97460631, 0.02539369],
       [0.94436786, 0.05563214],
       [0.69548987, 0.30451013],
       [0.79973024, 0.20026976],
       [0.90898503, 0.09101497],
       [0.42167972, 0.57832028],
       [0.04363903, 0.95636097],
       [0.02970679, 0.97029321],
       [0.54403301, 0.45596699],
       [0.88253385, 0.11746615],
       [0.95308941, 0.04691059],
       [0.55788172, 0.44211828],
       [0.11295424, 0.88704576],
       [0.80374304, 0.19625696],
       [0.84658318, 0.15341682],
       [0.33447657, 0.66552343],
       [0.73584701, 0.26415299],
       [0.54888918, 0.45111082],
       [0.15401224, 0.84598776],
       [0.86402358, 0.13597642],
       [0.17464805, 0.82535195],
       [0.09310965, 0.90689035],
       [0.92621473, 0.07378527],
       [0.16692787, 0.83307213],
```

```
[0.55238222, 0.44761778],
[0.9213803 , 0.0786197 ],
[0.09565109, 0.90434891],
[0.29087267, 0.70912733],
[0.6418648 , 0.3581352 ],
[0.34574597, 0.65425403],
[0.73907763, 0.26092237],
[0.74200067, 0.25799933],
[0.92247573, 0.07752427],
[0.45622019, 0.54377981],
[0.6616213 , 0.3383787 ],
[0.82258284, 0.17741716],
[0.76028762, 0.23971238],
[0.08045078, 0.91954922],
[0.51064148, 0.48935852],
[0.85316202, 0.14683798],
[0.85867315, 0.14132685],
[0.10768806, 0.89231194],
[0.34437339, 0.65562661],
[0.23330143, 0.76669857],
[0.88158729, 0.11841271],
[0.48917794, 0.51082206],
[0.84760955, 0.15239045],
[0.15813466, 0.84186534],
[0.16237414, 0.83762586],
[0.97068039, 0.02931961],
[0.39564656, 0.60435344],
[0.87500065, 0.12499935],
[0.47059133, 0.52940867],
[0.95105753, 0.04894247],
[0.93314053, 0.06685947],
[0.10968229, 0.89031771],
[0.31704496, 0.68295504],
[0.85114718, 0.14885282],
[0.47195419, 0.52804581],
[0.59428131, 0.40571869],
[0.77542573, 0.22457427],
[0.51686494, 0.48313506],
[0.73112616, 0.26887384],
[0.91674331, 0.08325669],
[0.35950129, 0.64049871],
[0.97583621, 0.02416379],
[0.26771362, 0.73228638],
[0.95741197, 0.04258803],
[0.72024255, 0.27975745],
[0.44512361, 0.55487639],
[0.81728835, 0.18271165],
[0.97231602, 0.02768398],
[0.56931915, 0.43068085],
[0.84124282, 0.15875718],
[0.61161179, 0.38838821],
[0.62392231, 0.37607769],
[0.2744716 , 0.7255284 ],
[0.70028481, 0.29971519],
[0.05311803, 0.94688197],
[0.85579789, 0.14420211],
[0.40386586, 0.59613414],
[0.95996188, 0.04003812],
[0.89506162, 0.10493838],
[0.981258  , 0.018742  ],
[0.01508682, 0.98491318],
[0.88742567, 0.11257433],
[0.60335826, 0.39664174],
[0.28210549, 0.71789451],
[0.11948805, 0.88051195],
[0.85714296, 0.14285704],
[0.44697938, 0.55302062],
[0.35039271, 0.64960729],
[0.60225937, 0.39774063],
```

```
       [0.8807046 , 0.1192954 ],
       [0.71208991, 0.28791009],
       [0.81950299, 0.18049701],
       [0.92182285, 0.07817715],
       [0.8404731 , 0.1595269 ],
       [0.87345451, 0.12654549],
       [0.08312523, 0.91687477],
       [0.9168552 , 0.0831448 ],
       [0.53966089, 0.46033911],
       [0.72197896, 0.27802104],
       [0.29818702, 0.70181298],
       [0.36246009, 0.63753991],
       [0.85182738, 0.14817262],
       [0.83578792, 0.16421208],
       [0.11990613, 0.88009387],
       [0.01581862, 0.98418138],
       [0.8146588 , 0.1853412 ],
       [0.97481847, 0.02518153],
       [0.97565869, 0.02434131],
       [0.93871749, 0.06128251],
       [0.94014738, 0.05985262],
       [0.95300419, 0.04699581],
       [0.7748833 , 0.2251167 ],
       [0.93668996, 0.06331004],
       [0.42012276, 0.57987724],
       [0.71912021, 0.28087979],
       [0.91966395, 0.08033605],
       [0.13537256, 0.86462744],
       [0.88827663, 0.11172337],
       [0.22298347, 0.77701653],
       [0.37397552, 0.62602448],
       [0.02105151, 0.97894849],
       [0.66452579, 0.33547421],
       [0.70422297, 0.29577703],
       [0.46594125, 0.53405875],
       [0.37430712, 0.62569288],
       [0.66899373, 0.33100627],
       [0.92088645, 0.07911355],
       [0.93277213, 0.06722787],
       [0.95894943, 0.04105057],
       [0.69555   , 0.30445   ],
       [0.24442913, 0.75557087],
       [0.70753602, 0.29246398],
       [0.9218604 , 0.0781396 ],
       [0.68400304, 0.31599696],
       [0.57490655, 0.42509345],
       [0.1557367 , 0.8442633 ],
       [0.8690276 , 0.1309724 ],
       [0.96207402, 0.03792598],
       [0.86738339, 0.13261661],
       [0.55994202, 0.44005798],
       [0.45147594, 0.54852406],
       [0.98601905, 0.01398095],
       [0.6031482 , 0.3968518 ],
       [0.98772617, 0.01227383],
       [0.26209536, 0.73790464],
       [0.33027675, 0.66972325],
       [0.71621322, 0.28378678],
       [0.029164  , 0.970836  ],
       [0.34016187, 0.65983813],
       [0.90937641, 0.09062359],
       [0.5966371 , 0.4033629 ],
       [0.98653749, 0.01346251],
       [0.2057542 , 0.7942458 ]])
```

In [287…

```
confusion_matrix(y_test,prediction)
```

Out[287… array([[103,  19],

```
        [ 26,  44]], dtype=int64)
```

In [288…

```
classification_report(y_test,prediction)
```

Out[288…

```
'              precision    recall  f1-score   support\n\n           0       0.80
0.84      0.82       122\n           1       0.70      0.63      0.66       70\n\n
accuracy                           0.77       192\n   macro avg       0.75      0.74
0.74       192\nweighted avg       0.76      0.77      0.76       192\n'
```

In [289…

```
r2_score(y_test,prediction)
```

Out[289…

```
-0.011709601873536090
```

In [290…

```
log_reg.coef_
```

Out[290…

```
array([[ 1.88860124e-01,  3.09970727e-02, -2.96395017e-02,
        -2.35263940e-02, -2.15472324e-04,  1.37493724e-01,
         1.66425827e+00,  2.96492011e-03]])
```

In [291…

```
log_reg.intercept_
```

Out[291…

```
array([-7.85430848])
```

In [292…

```
print(classification_report(y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       0.80      0.84      0.82       122
           1       0.70      0.63      0.66        70

    accuracy                           0.77       192
   macro avg       0.75      0.74      0.74       192
weighted avg       0.76      0.77      0.76       192
```

In [293…

```
accuracy_score(y_test,prediction)
```

Out[293…

```
0.765625
```

In [294…

```
#ROC curve ,AUC

from sklearn.metrics import roc_curve,auc,roc_auc_score
```

In [295…

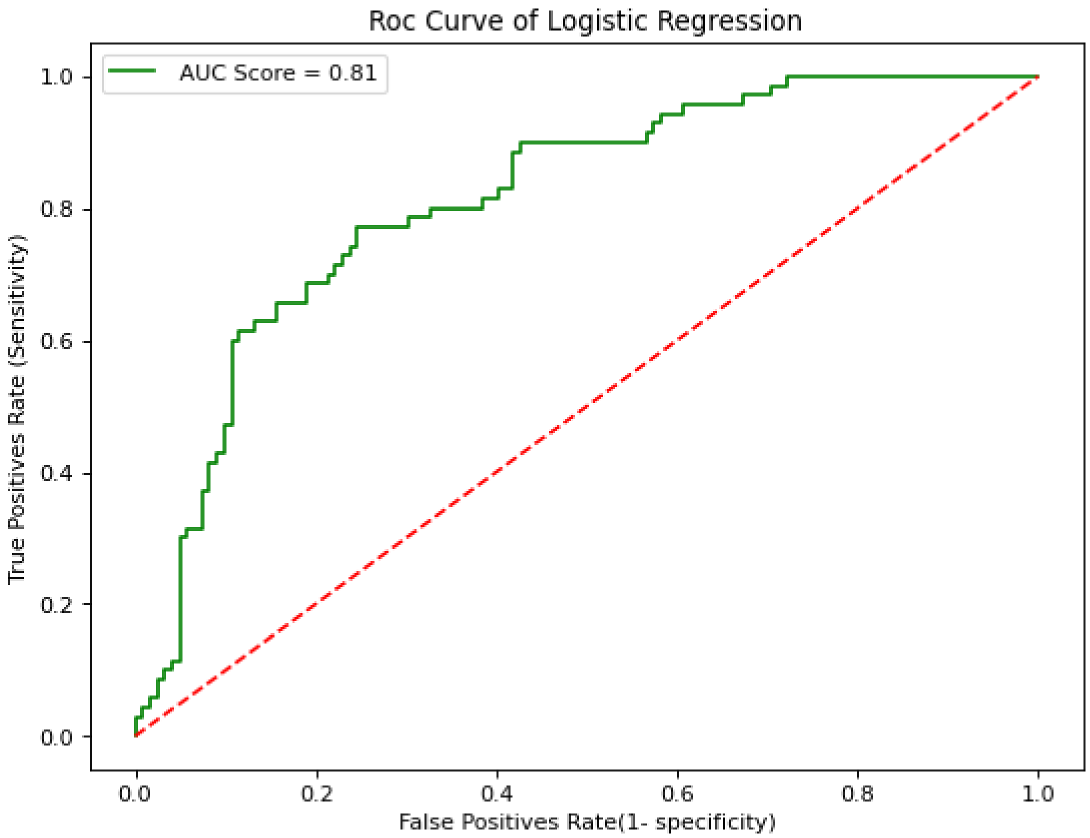
```
fp_logistic , tp_logistic ,threshold_logistc = roc_curve(y_test,log_reg.predict_prob
auc_logistic = auc(fp_logistic,tp_logistic)
```

In [296…

```
print("Logostic Model:")
print("Accuracy score of Logistic Regression Model :: ",accuracy_score(y_test,predic
print("Classification Report :")
print(classification_report(y_test,prediction))
print("ROC Curve")

plt.figure(figsize=(8,6),dpi=80)
plt.title("Roc Curve of Logistic Regression")
plt.plot(fp_logistic,tp_logistic,'b',label = " AUC Score = %0.2f"%auc_logistic , col
```

```
plt.plot(fp_logistic,fp_logistic,'r--',color="red")

plt.xlabel("False Positives Rate(1- specificity)")
plt.ylabel("True Positives Rate (Sensitivity)")

plt.legend()
plt.show()
```

```
Logostic Model:
Accuracy score of Logistic Regression Model ::  0.765625
Classification Report :
              precision    recall  f1-score   support

           0       0.80      0.84      0.82       122
           1       0.70      0.63      0.66        70

    accuracy                           0.77       192
   macro avg       0.75      0.74      0.74       192
weighted avg       0.76      0.77      0.76       192

ROC Curve
```




## Accuracy Score for Logistic Algorithm : 76.5%

# KNN Algorithm

In [297…

```
from sklearn.neighbors import KNeighborsClassifier

knc = KNeighborsClassifier( n_neighbors=1) #  n_neighbours = 1
```

In [298…

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

Out[298…

```
KNeighborsClassifier(n_neighbors=1)
```

In [299…
```
knc.score(x_train,y_train)
```

Out[299… 1.0

In [300…
```
knc.score(x_test,y_test)
```

Out[300… 0.6666666666666666

In [301…
```
# difference = 31 % , lets see below
knc.score(x_train,y_train) - knc.score(x_test,y_test)
```

Out[301… 0.33333333333333337

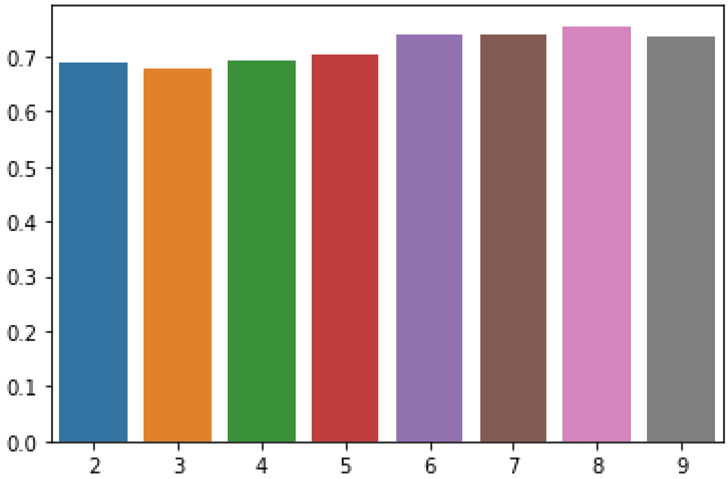
In [302…
```
# checking for n neighbours

neighbours = [i for i in range(2,10)]
accuracies = []

for i in neighbours:
    knc = KNeighborsClassifier(n_neighbors=i)
    knc.fit(x_train,y_train)
    accuracies.append(knc.score(x_test,y_test))
```

In [303…
```
print("accuracy of k = 2 is ",accuracies[0])
print("accuracy of k = 3 is ",accuracies[1])
print("accuracy of k = 4 is ",accuracies[2])
print("accuracy of k = 5 is ",accuracies[3])
print("accuracy of k = 6 is ",accuracies[4])
print("accuracy of k = 7 is ",accuracies[5])
print("accuracy of k = 8 is ",accuracies[6])
print("accuracy of k = 9 is ",accuracies[7])
```

```
accuracy of k = 2 is  0.6875
accuracy of k = 3 is  0.6770833333333334
accuracy of k = 4 is  0.6927083333333334
accuracy of k = 5 is  0.703125
accuracy of k = 6 is  0.7395833333333334
accuracy of k = 7 is  0.7395833333333334
accuracy of k = 8 is  0.7552083333333334
accuracy of k = 9 is  0.734375
```

In [304…
```
## plotting bargraph between neighbours(k) and test accuracies

sns.barplot(x=  neighbours, y= accuracies)
```

Out[304… <AxesSubplot:>




In [305…
```
## I can see k = 9 gives better accuracy

knc = KNeighborsClassifier(n_neighbors=8)
```

In [306…
```
knc.fit(x_train,y_train)
```

Out[306… KNeighborsClassifier(n_neighbors=8)

In [307…
```
knc.score(x_train,y_train)
```

Out[307… 0.8038194444444444

In [308…
```
knc.score(x_test,y_test) # now the model is in good condition
```

Out[308… 0.7552083333333334

In [309…
```
knc_prediction = knc.predict(x_test)
```

In [310…
```
accuracy_score(y_test,knc_prediction)
```

Out[310… 0.7552083333333334

In [311…
```
print(classification_report(y_test,knc_prediction))
```

```
              precision    recall  f1-score   support

           0       0.77      0.89      0.82       122
           1       0.73      0.53      0.61        70

    accuracy                           0.76       192
   macro avg       0.75      0.71      0.72       192
weighted avg       0.75      0.76      0.74       192
```

In [312…
```
confusion_matrix(y_test,knc_prediction)
```

Out[312…
```
array([[108,  14],
       [ 33,  37]], dtype=int64)
```

In [313…

```
r2_score(y_test,knc_prediction)
```

Out[313…

```
-0.05667447306791562
```

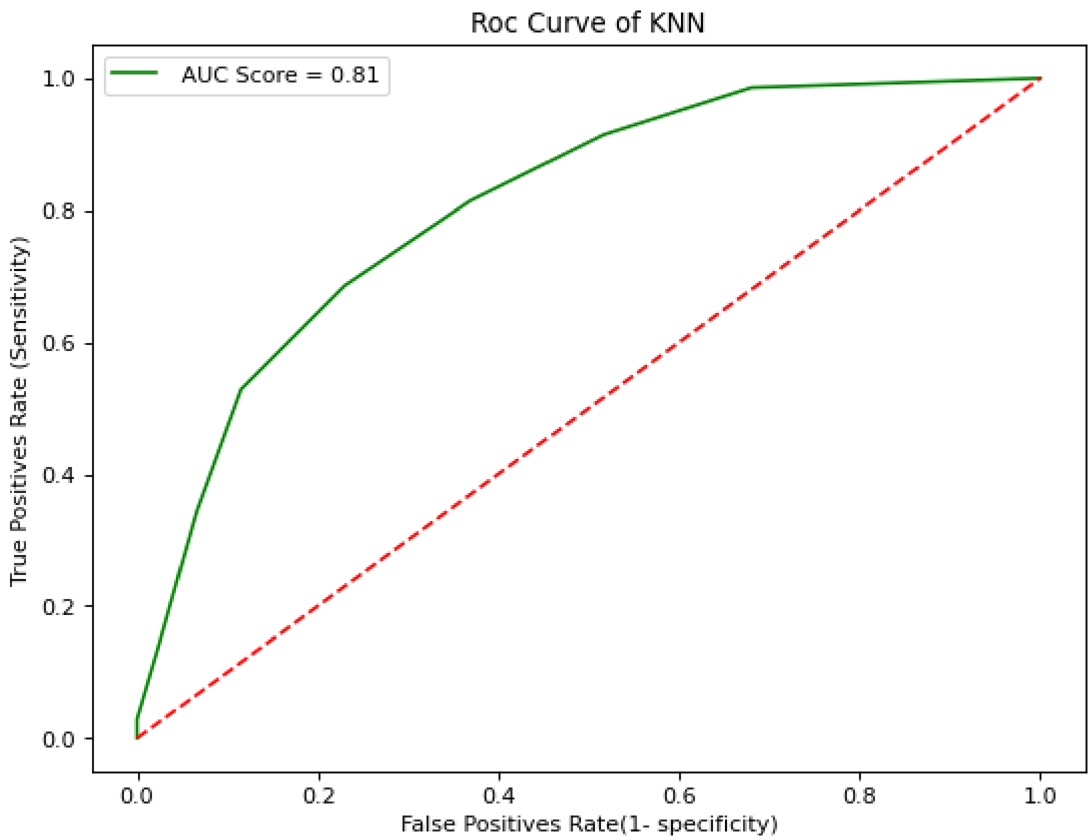
In [314…

```
# roc-curve,auc

fp_knn , tp_knn,threshold_knn = roc_curve(y_test,knc.predict_proba(x_test)[:,1])

auc_knn = auc(fp_knn,tp_knn)
```

In [315…

```
print("KNN Model:")
print("Accuracy score of KNN Model :: ",accuracy_score(y_test,knc_prediction))
print("Classification Report :")
print(classification_report(y_test,knc_prediction))
print("ROC Curve")

plt.figure(figsize=(8,6),dpi=80)
plt.title("Roc Curve of KNN")
plt.plot(fp_knn,tp_knn,'b',label = " AUC Score = %0.2f"%auc_knn, color = "green")
plt.plot(fp_knn,fp_knn,'r--',color="red")

plt.xlabel("False Positives Rate(1- specificity)")
plt.ylabel("True Positives Rate (Sensitivity)")

plt.legend()
plt.show()
```

```
KNN Model:
Accuracy score of KNN Model ::  0.7552083333333334
Classification Report :
              precision    recall  f1-score   support

           0       0.77      0.89      0.82       122
           1       0.73      0.53      0.61        70

    accuracy                           0.76       192
   macro avg       0.75      0.71      0.72       192
weighted avg       0.75      0.76      0.74       192

ROC Curve
```




# Acuracy score for KNN : 75.5%

Which is not better than Logistic model

# Decision Tree Classifier Model

In [316…
```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
```

In [317…
```
dtc.fit(x_train,y_train)
```

Out[317… DecisionTreeClassifier()

In [318…
```
dtc.score(x_train,y_train)
```

Out[318… 1.0

In [319…
```
dtc.score(x_test,y_test)
```

Out[319… 0.7291666666666666

In [320…
```
 #difference = 30% , huge it is underfited
dtc.score(x_train,y_train) -  dtc.score(x_test,y_test)
```

Out[320… 0.27083333333333337

In [321…
```
dtc_prediction = dtc.predict(x_test)
```

In [322…
```
accuracy_score(y_test,dtc_prediction)
```

Out[322…
```
0.7291666666666666
```

In [323…
```
dtc_train_pred = dtc.predict(x_train)
```

In [324…
```
accuracy_score(y_train,dtc_train_pred)
```

Out[324…
```
1.0
```

In [325…
```
from sklearn.model_selection import GridSearchCV

param_grid = { 'max_depth':[1,2,3,4,5,None],'min_samples_leaf':[1,2,3,4]}
```

In [326…
```
gs = GridSearchCV(dtc,param_grid=param_grid,cv = 3,verbose=1)
```

In [327…
```
gs.fit(X,y)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

Out[327…
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5, None],
                         'min_samples_leaf': [1, 2, 3, 4]},
             verbose=1)
```

In [328…
```
gs.best_params_
```

Out[328…
```
{'max_depth': 4, 'min_samples_leaf': 3}
```

In [329…
```
gs.best_score_
```

Out[329…
```
0.7369791666666666
```

In [330…
```
dtc.feature_importances_
```

Out[330…
```
array([0.04594997, 0.28948875, 0.07407016, 0.07575794, 0.06176781,
       0.17541518, 0.1229738 , 0.15457638])
```

In [331…
```
dtc = DecisionTreeClassifier(max_depth=4,min_samples_leaf=3)
```

In [332…
```
dtc.fit(x_train,y_train)
```

Out[332…
```
DecisionTreeClassifier(max_depth=4, min_samples_leaf=3)
```

In [333…
```
dtc.score(x_train,y_train)
```

Out[333…
```
0.8159722222222222
```

In [334…
```
dtc.score(x_test,y_test)
```
Out[334… 0.7552083333333334

In [335…
```
dtc_text_pred = dtc.predict(x_test)
```

In [336…
```
accuracy_score(y_test,dtc_text_pred)
```
Out[336… 0.7552083333333334

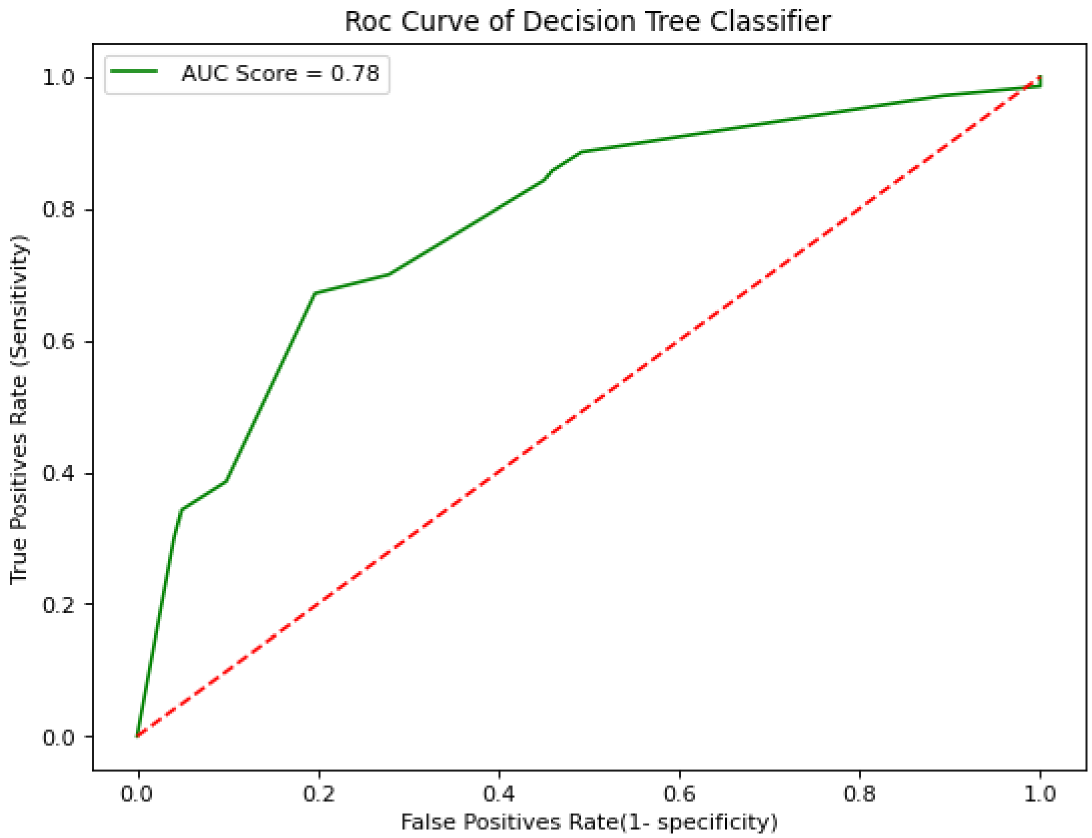
In [337…
```
print(classification_report(y_test,dtc_text_pred))
```
```
              precision    recall  f1-score   support

           0       0.81      0.80      0.81       122
           1       0.66      0.67      0.67        70

    accuracy                           0.76       192
   macro avg       0.74      0.74      0.74       192
weighted avg       0.76      0.76      0.76       192
```

In [338…
```
confusion_matrix(y_test,dtc_text_pred)
```
Out[338…
```
array([[98, 24],
       [23, 47]], dtype=int64)
```

In [339…
```
# roc-curve , auc

fp_dtc,tp_dtc ,threshold = roc_curve(y_test,dtc.predict_proba(x_test)[:,1])
auc_dtc = auc(fp_dtc,tp_dtc)
```

In [340…
```
print("Decision Tree Classifier Model:")
print("Accuracy score of Decision Tree Model :: ",accuracy_score(y_test,dtc_text_pre
print("Classification Report :")
print(classification_report(y_test,dtc_text_pred))
print("ROC Curve")

plt.figure(figsize=(8,6),dpi=80)
plt.title("Roc Curve of Decision Tree Classifier")
plt.plot(fp_dtc,tp_dtc,'b',label = " AUC Score = %0.2f"%auc_dtc , color = "green")
plt.plot(fp_dtc,fp_dtc,'r--',color="red")

plt.xlabel("False Positives Rate(1- specificity)")
plt.ylabel("True Positives Rate (Sensitivity)")

plt.legend()
plt.show()
```
```
Decision Tree Classifier Model:
Accuracy score of Decision Tree Model ::  0.7552083333333334
Classification Report :
              precision    recall  f1-score   support

           0       0.81      0.80      0.81       122
           1       0.66      0.67      0.67        70
```

```
    accuracy                           0.76       192
   macro avg       0.74      0.74      0.74       192
weighted avg       0.76      0.76      0.76       192

ROC Curve
```




# Accuracy score for Decision Tree classifier : 75.5%

# Random Forest Classifier

In [341…

```python
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()
```

In [342…

```python
rfc.fit(x_train,y_train)
```

Out[342… RandomForestClassifier()

In [343…

```python
rfc.score(x_train,y_train)
```

Out[343… 1.0

In [344…

```python
rfc.score(x_test,y_test)
```

Out[344… 0.7604166666666666

In [345…

```python
# the difference b/w scores of train and test is 19% too large , going with gridsear

rfc.score(x_train,y_train) - rfc.score(x_test,y_test)
```

Out[345… 0.23958333333333337

In [346…
```
param_grid_rfc = {'n_estimators':[150,200,250],'max_depth':[None , 1 ,3, 5],'min_sam
```

In [347…
```
gs_rfc = GridSearchCV(estimator=rfc,param_grid=param_grid_rfc,cv=3,verbose=1)
```

In [348…
```
gs_rfc.fit(X,y)
```

```
Fitting 3 folds for each of 60 candidates, totalling 180 fits
```

Out[348…
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [None, 1, 3, 5],
                         'min_samples_leaf': [1, 3, 5, 7, 9],
                         'n_estimators': [150, 200, 250]},
             verbose=1)
```

In [349…
```
gs_rfc.best_params_
```

Out[349… {'max_depth': None, 'min_samples_leaf': 3, 'n_estimators': 200}

In [351…
```
rfc = RandomForestClassifier(max_depth=None,min_samples_leaf=3,n_estimators = 200)
```

In [352…
```
rfc.fit(x_train,y_train)
```

Out[352… RandomForestClassifier(min_samples_leaf=3, n_estimators=200)

In [353…
```
rfc.score(x_train,y_train)
```

Out[353… 0.953125

In [354…
```
rfc.score(x_test,y_test)
```

Out[354… 0.78125

In [355…
```
rfc_prediction = rfc.predict(x_test)
```

In [356…
```
accuracy_score(y_test,rfc_prediction)
```

Out[356… 0.78125

In [357…
```
print(classification_report(y_test,rfc_prediction))
```

```
              precision    recall  f1-score   support

           0       0.81      0.85      0.83       122
           1       0.72      0.66      0.69        70

    accuracy                           0.78       192
   macro avg       0.77      0.75      0.76       192
weighted avg       0.78      0.78      0.78       192
```

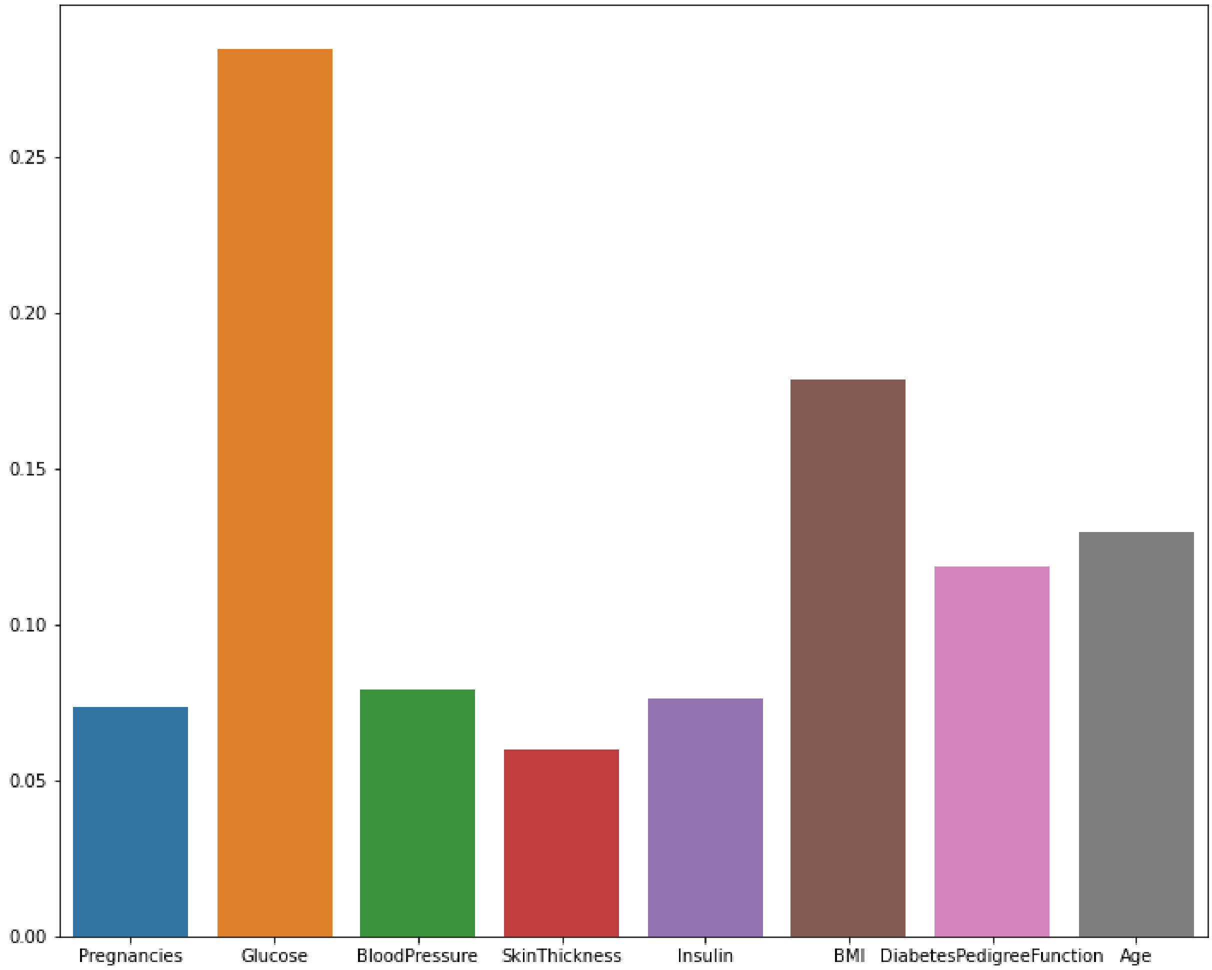
In [358…
```
confusion_matrix(y_test,rfc_prediction)
```

Out[358…
```
array([[104,  18],
       [ 24,  46]], dtype=int64)
```

In [359…
```
rfc.feature_importances_
```

Out[359…
```
array([0.07359505, 0.28466684, 0.07881504, 0.05958517, 0.07634564,
       0.1784295 , 0.11871575, 0.12984701])
```

In [360…
```
#  plotting barplot b/w x_train.columns and rfc.feature_importances_

plt.figure(figsize=(12,10))
sns.barplot(x= x_train.columns , y=rfc.feature_importances_)
plt.show()
```

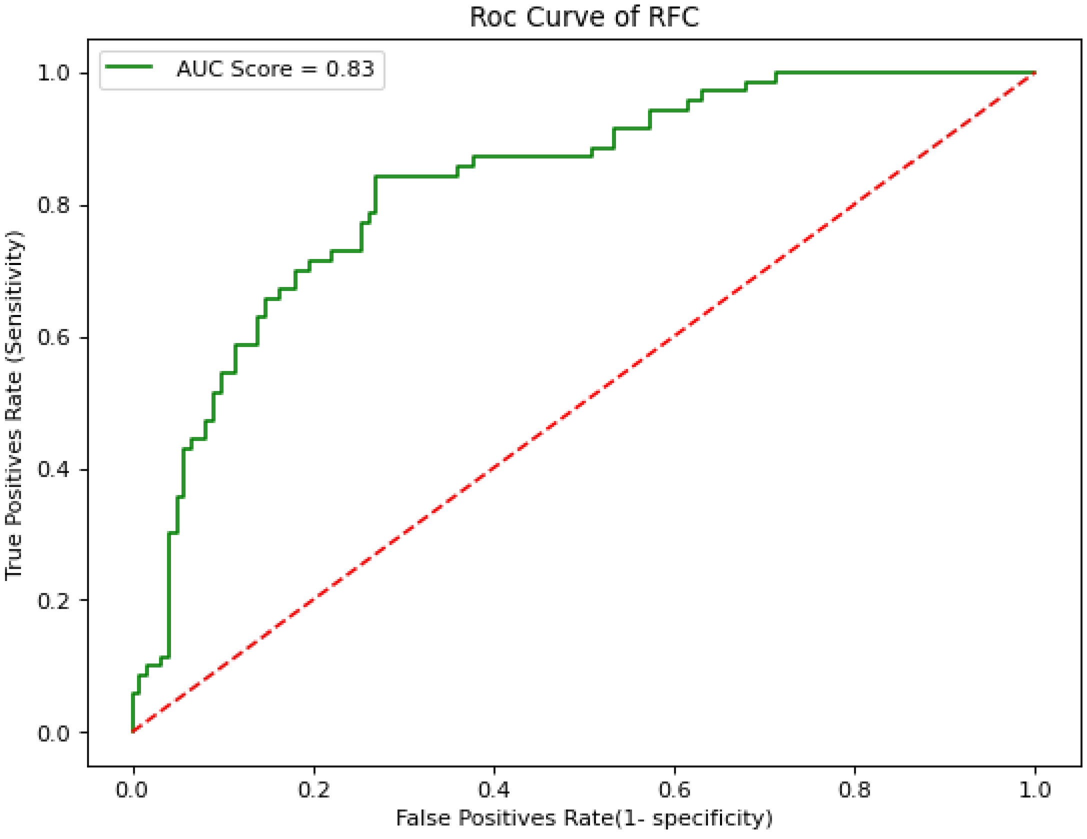



In [361…
```
# roc-curve , auc

fp_rfc,tp_rfc,threshold_rfc = roc_curve(y_test,rfc.predict_proba(x_test)[:,1])
auc_rfc = auc(fp_rfc,tp_rfc)
```

In [362…
```
print("Random Forest Classifier Model:")
print("Accuracy score of RFC Model :: ",accuracy_score(y_test,rfc_prediction))
print("Classification Report :")
print(classification_report(y_test,rfc_prediction))
print("ROC Curve")

plt.figure(figsize=(8,6),dpi=80)
```

```
plt.title("Roc Curve of RFC")
plt.plot(fp_rfc,tp_rfc,'b',label = " AUC Score = %0.2f"%auc_rfc , color = "green")
plt.plot(fp_rfc,fp_rfc,'r--',color="red")

plt.xlabel("False Positives Rate(1- specificity)")
plt.ylabel("True Positives Rate (Sensitivity)")

plt.legend()
plt.show()
```

```
Random Forest Classifier Model:
Accuracy score of RFC Model ::  0.78125
Classification Report :
              precision    recall  f1-score   support

           0       0.81      0.85      0.83       122
           1       0.72      0.66      0.69        70

    accuracy                           0.78       192
   macro avg       0.77      0.75      0.76       192
weighted avg       0.78      0.78      0.78       192

ROC Curve
```




## Accuracy score for Random Forest Classifier : 78.1%

## Support Vector Machines(SVM)

In [363...

```
from sklearn.svm import SVC
svc = SVC()
```

In [364...

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

Out[364... SVC()

In [365…
```
svc.score(x_train,y_train)
```

Out[365…
```
0.7586805555555556
```

In [366…
```
svc.score(x_test,y_test)
```

Out[366…
```
0.7708333333333334
```

In [367…
```
# normally fitted
svc.score(x_train,y_train)-svc.score(x_test,y_test)
```

Out[367…
```
-0.012152777777777779
```

In [368…
```
# to check the best parameters of svc

param_grid = { 'C':[0.1,1,5,10],'gamma':[0.001,0.0001,0.00001,0.000001]}
```

In [369…
```
gs_svc = GridSearchCV(estimator= svc, param_grid= param_grid,cv=3,verbose=1)
```

In [370…
```
gs_svc.fit(X,y)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
```

Out[370…
```
GridSearchCV(cv=3, estimator=SVC(),
             param_grid={'C': [0.1, 1, 5, 10],
                         'gamma': [0.001, 0.0001, 1e-05, 1e-06]},
             verbose=1)
```

In [371…
```
gs_svc.best_params_
```

Out[371…
```
{'C': 1, 'gamma': 0.0001}
```

In [372…
```
gs_svc.score
```

Out[372…
```
<bound method BaseSearchCV.score of GridSearchCV(cv=3, estimator=SVC(),
             param_grid={'C': [0.1, 1, 5, 10],
                         'gamma': [0.001, 0.0001, 1e-05, 1e-06]},
             verbose=1)>
```

In [373…
```
svc = SVC(C=1,gamma=0.0001 , probability=True)
```

In [374…
```
svc.fit(x_train,y_train)
```

Out[374…
```
SVC(C=1, gamma=0.0001, probability=True)
```

In [375…
```
svc.score(x_train,y_train)
```

Out[375…
```
0.7690972222222222
```

In [376…
```
svc.score(x_test,y_test)
```

Out[376… 0.765625

In [377…
```
svc_pred = svc.predict(x_test)
```

In [378…
```
accuracy_score(y_test,svc_pred)
```

Out[378… 0.765625

In [379…
```
print(classification_report(y_test,svc_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.93      0.83       122
           1       0.79      0.49      0.60        70

    accuracy                           0.77       192
   macro avg       0.77      0.71      0.72       192
weighted avg       0.77      0.77      0.75       192
```

In [380…
```
confusion_matrix(y_test,svc_pred)
```

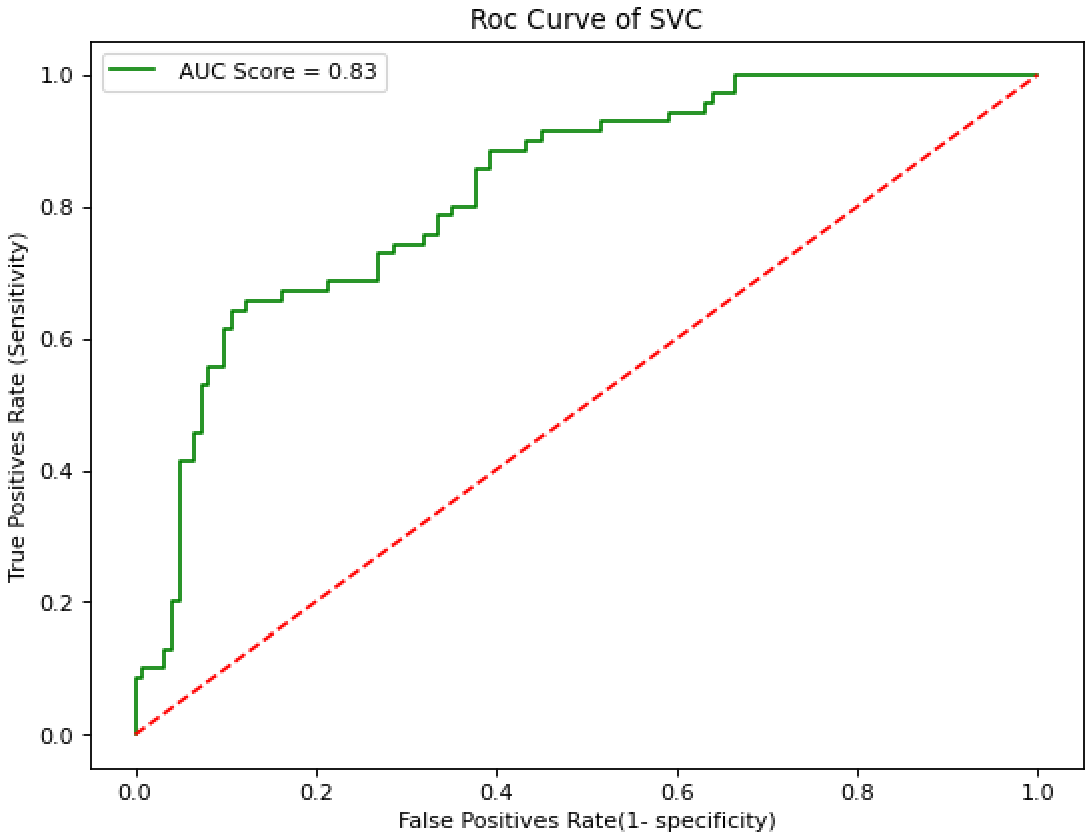
Out[380…
```
array([[113,   9],
       [ 36,  34]], dtype=int64)
```

In [381…
```
# roc-curve , auc

fp_svc, tp_svc , threshold_svc = roc_curve(y_test,svc.predict_proba(x_test)[:,1])
auc_svc = auc(fp_svc,tp_svc)
```

In [382…
```
print("Support Vector Classifier  Model:")
print("Accuracy score of SVC Model :: ",accuracy_score(y_test,svc_pred))
print("Classification Report :")
print(classification_report(y_test,svc_pred))
print("ROC Curve")

plt.figure(figsize=(8,6),dpi=80)
plt.title("Roc Curve of SVC")
plt.plot(fp_svc,tp_svc,'b',label = " AUC Score = %0.2f"%auc_svc , color = "green")
plt.plot(fp_svc,fp_svc,'r--',color="red")

plt.xlabel("False Positives Rate(1- specificity)")
plt.ylabel("True Positives Rate (Sensitivity)")

plt.legend()
plt.show()
```

```
Support Vector Classifier  Model:
Accuracy score of SVC Model ::  0.765625
Classification Report :
              precision    recall  f1-score   support

           0       0.76      0.93      0.83       122
           1       0.79      0.49      0.60        70

    accuracy                           0.77       192
   macro avg       0.77      0.71      0.72       192
weighted avg       0.77      0.77      0.75       192
```

ROC Curve




# Accuracy score for SVM :76.5%

In [383…

```
print("Accuracy score of Logistic Regression Model :: ",accuracy_score(y_test,predic
print("Accuracy score of KNN Model :: ",accuracy_score(y_test,knc_prediction))
print("Accuracy score of Decision Tree Model :: ",accuracy_score(y_test,dtc_text_pre
print("Accuracy score of RFC Model :: ",accuracy_score(y_test,rfc_prediction))
print("Accuracy score of SVC Model :: ",accuracy_score(y_test,svc_pred))
```

```
Accuracy score of Logistic Regression Model ::  0.765625
Accuracy score of KNN Model ::  0.7552083333333334
Accuracy score of Decision Tree Model ::  0.7552083333333334
Accuracy score of RFC Model ::  0.78125
Accuracy score of SVC Model ::  0.765625
```

In [384…

```
print("Classification Report of Logistic :")
print(classification_report(y_test,prediction))
print("Classification Report of KNN :")
print(classification_report(y_test,knc_prediction))
print("Classification Report of Decision tree :")
print(classification_report(y_test,dtc_text_pred))
print("Classification Report of RFC :")
print(classification_report(y_test,rfc_prediction))
print("Classification Report of SVC :")
print(classification_report(y_test,svc_pred))
```

```
Classification Report of Logistic :
              precision    recall  f1-score   support

           0       0.80      0.84      0.82       122
           1       0.70      0.63      0.66        70

    accuracy                           0.77       192
   macro avg       0.75      0.74      0.74       192
```

```
weighted avg       0.76      0.77      0.76       192

Classification Report of KNN :
              precision    recall  f1-score   support

           0       0.77      0.89      0.82       122
           1       0.73      0.53      0.61        70

    accuracy                           0.76       192
   macro avg       0.75      0.71      0.72       192
weighted avg       0.75      0.76      0.74       192

Classification Report of Decision tree :
              precision    recall  f1-score   support

           0       0.81      0.80      0.81       122
           1       0.66      0.67      0.67        70

    accuracy                           0.76       192
   macro avg       0.74      0.74      0.74       192
weighted avg       0.76      0.76      0.76       192

Classification Report of RFC :
              precision    recall  f1-score   support

           0       0.81      0.85      0.83       122
           1       0.72      0.66      0.69        70

    accuracy                           0.78       192
   macro avg       0.77      0.75      0.76       192
weighted avg       0.78      0.78      0.78       192

Classification Report of SVC :
              precision    recall  f1-score   support

           0       0.76      0.93      0.83       122
           1       0.79      0.49      0.60        70

    accuracy                           0.77       192
   macro avg       0.77      0.71      0.72       192
weighted avg       0.77      0.77      0.75       192
```

### When compare other models with KNN Algorithm , I can say that Random Forest is better.

### By analysing the accuracy scores for different models, I am concluding that Random Forest classifier gives best accuarcy among other models

### I am considering it to be best because balance of class between Precision and Recall better then other models.

In [ ]:

In [ ]:

In [ ]: