Data Science Capstone

Project 2 - Healthcare

Problem Statement

- Build a model to accurately predict whether the patients in the dataset have diabetes or not.

In [1]:
```
#importing necessary libraries
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
#importing libraries for visualisation
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
```

In [2]:
```
#importing Data
data_file=r'C:\Users\sinun\OneDrive\Documents\Simplilearn\Capstone\project\Project 2\Healthcare - Diabetes\health care diabetes.cs
data_frame=pd.read_csv(data_file)
```

Project Task: Week 1

Data Exploration:

1) Perform descriptive analysis. Understand the variables and their corresponding values.

In [3]:
```
# Understanding the Data Variables
data_frame.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 [4]:

```
# Show the top 5 Rows of data
data_frame.head()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	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 [5]:

```
# Performing Descriptive Analysis
data_frame.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00

	count	mean	std	min	25%	50%	75%	max
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

In [6]:
```
# Displaying the columns in the dataset
data_frame.columns
```

Out[6]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [7]:
```
# Checking for null values
data_frame.isnull().sum()
```

Out[7]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

- No missing values in any columns

In [8]:
```
# On the columns Glucose, BloodPressure, SkinThickness, Insulin and BMI, a value of zero does not make sense and thus indicates mi
# Explore more about the values and their count in these columns
imp_cols=['Glucose','BloodPressure','SkinThickness','BMI','Insulin']
def number_count(): # function defined for finding value counts in respective columns
    for col in imp_cols:
        print('Name of Variable :', col)
```

```
        print(data_frame[col].value_counts(),'\n\n')
number_count() # Function Called
```

```
Name of Variable : Glucose
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


Name of Variable : BloodPressure
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


Name of Variable : SkinThickness
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


Name of Variable : BMI
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


Name of Variable : Insulin
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
```

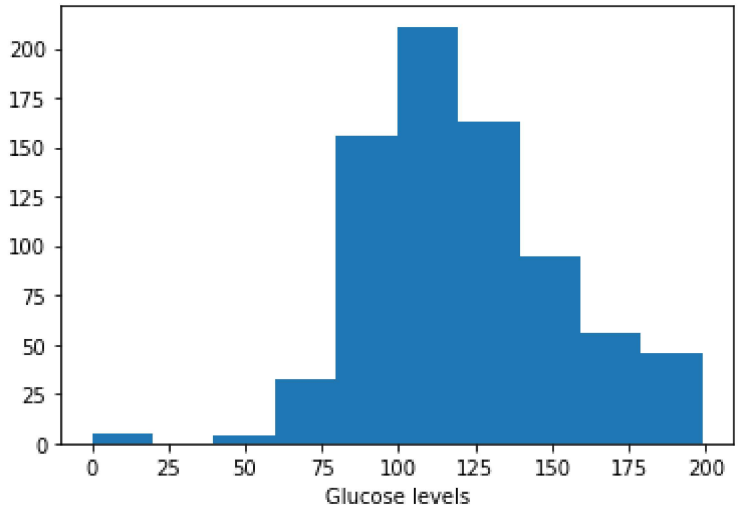
- By observing the above data we can understand that there are 35 values in Blood Pressure, 374 values in Insulin, 11 values in BMI, 227 values in Skin Thickness that are of '0' value and thus indicates missing values

2) Visually explore variables using histograms and treat the missing values accordingly

In [9]:

```
# Plot the histogram of the variable 'Glucose'
plt.hist(data_frame['Glucose'])
plt.xlabel('Glucose levels')
# Find mean of variable
print("Mean of Glucose level is :", data_frame['Glucose'].mean())
```

```
Mean of Glucose level is : 120.89453125
```

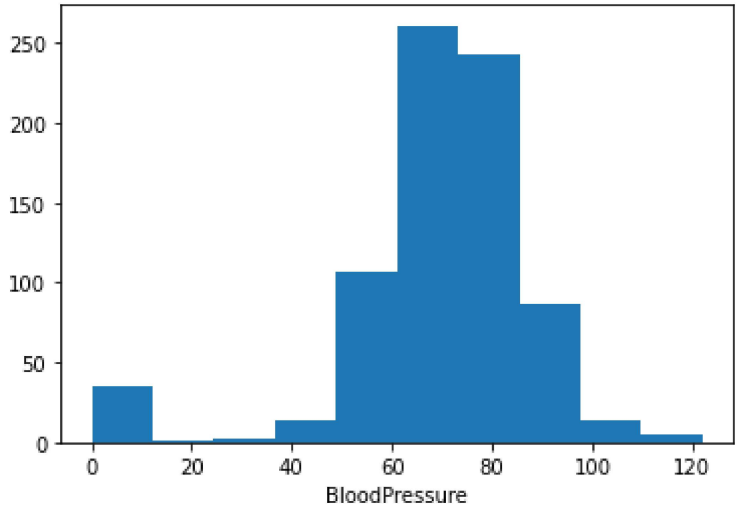


Most of the 'Glucose' Data is ranging between 100 and 140

In [10]:

```
# Plot the histogram of the variable 'BloodPressure'
plt.hist(data_frame['BloodPressure'])
plt.xlabel('BloodPressure')
# Find mean of variable
print("Mean of BloodPressure is :", data_frame['BloodPressure'].mean())
```

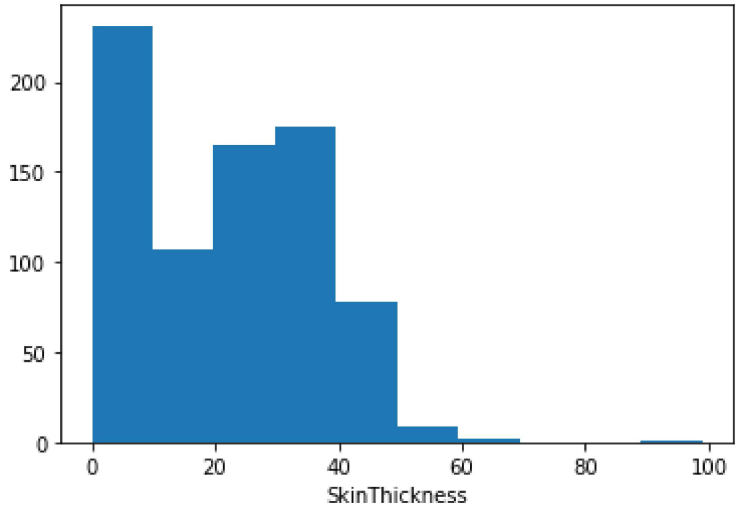
```
Mean of BloodPressure is : 69.10546875
```



Most of the data in 'BloodPressure' variable ranges between 60 and 90

In [11]:
```
# Plot the histogram of the variable 'SkinThickness'
plt.hist(data_frame['SkinThickness'])
plt.xlabel('SkinThickness')
# Find mean of variable
print("Mean of SkinThickness is :", data_frame['SkinThickness'].mean())
print("Datatype of SkinThickness Variable is:",data_frame['SkinThickness'].dtypes)
```

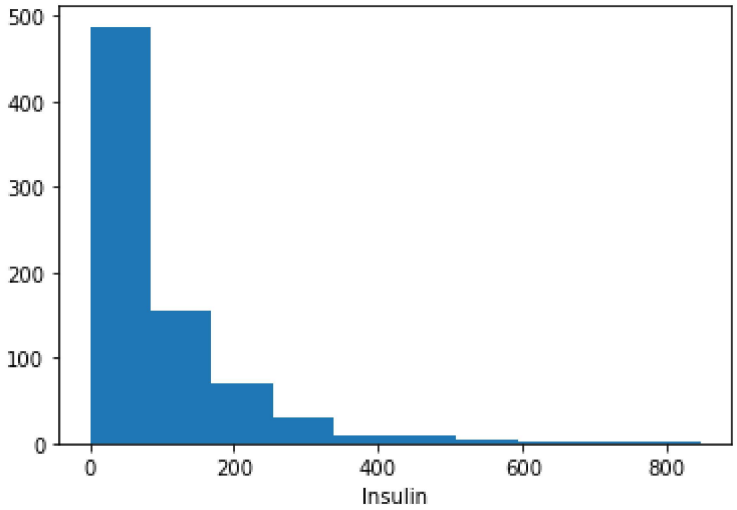
```
Mean of SkinThickness is : 20.536458333333332
Datatype of SkinThickness Variable is: int64
```



Most of the data in 'SkinThickness' variable ranges between 0 and 40

In [12]:
```
# Plot the histogram of the variable
plt.hist(data_frame['Insulin'])
plt.xlabel('Insulin')
# Find mean of variable
print("Mean of Insulin is :", data_frame['Insulin'].mean())
```

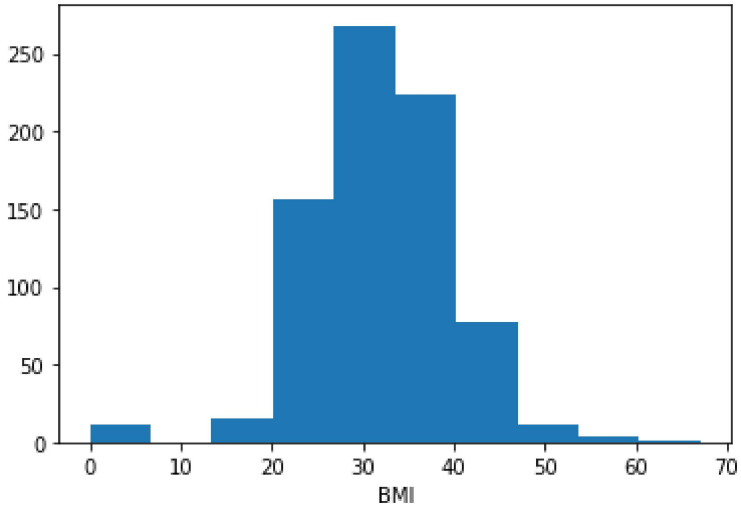
```
Mean of Insulin is : 79.79947916666667
```



In [13]:

```
# Plot the histogram of the variable BMI
plt.hist(data_frame['BMI'])
plt.xlabel('BMI')
# Find mean of variable
print("Mean of BMI is :", data_frame['BMI'].mean())
```

```
Mean of BMI is : 31.992578124999977
```



In [14]:

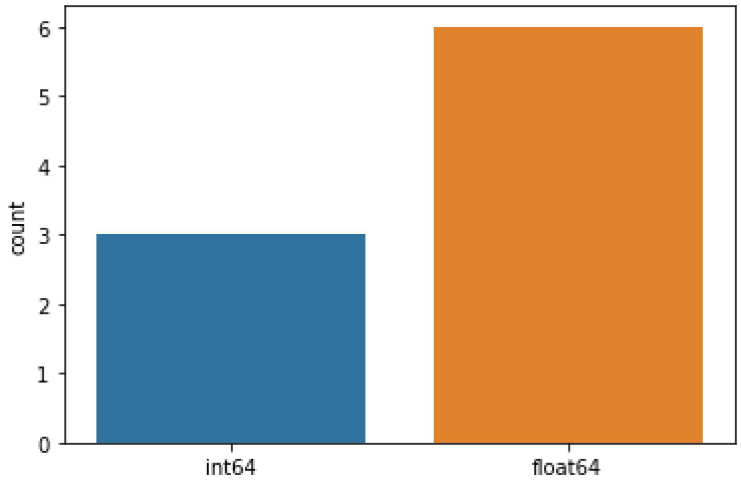
```
# On the columns Glucose, BloodPressure, SkinThickness, Insulin and BMI, a value of zero does not make sense and thus indicates mi
```

```
# Replace those zero values if any with the mean of the respective columns
imp_cols=['Glucose','BloodPressure','SkinThickness','BMI','Insulin']
def replace_zero_with_mean(): # Function defined for replacing zero values in columns with their mean value
    for col in imp_cols:
        data_frame[col]=data_frame[col].replace(0,data_frame[col].mean())

replace_zero_with_mean() # function called
```

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.

In [15]:

```
# create count plot describing the data types and the count of variables.
sns.countplot(data_frame.dtypes.map(str)) #  convert the type objects to string for plotting.
plt.show()
```



Project Task: Week 2

Data Exploration:

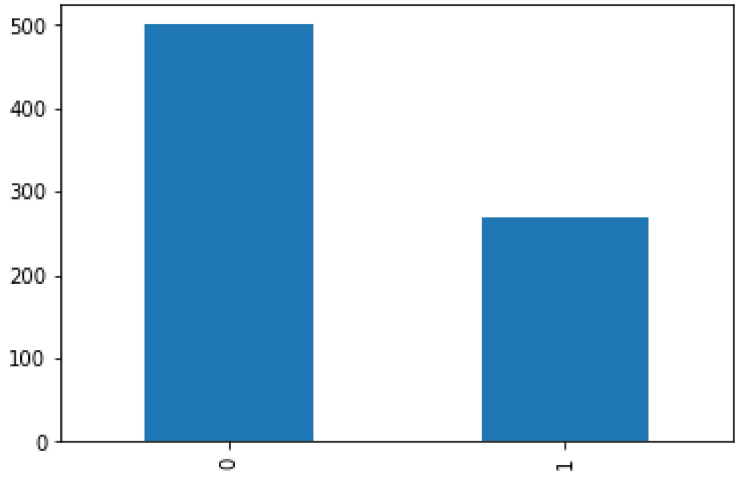
1) Check the balance of the data by plotting the count of outcomes by their value. Describe findings and plan future course of action.

In [16]:
```
##Checking the differnt values and their count in 'Outcome' variable
data_frame['Outcome'].value_counts()
```

Out[16]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [17]:
```
# Check for any data imbalance by plotting the count of outcomes by the values
data_frame['Outcome'].value_counts().plot(kind='bar')
```

Out[17]:
```
<AxesSubplot:>
```



- Majority Class (Class 0 or non- Diabetic) percentage = (500/768)* 100 = 65 %
- Minority Class (Class 1 or Diabetic) percentage = (268/768)* 100 = 35 %

Here dataset is SLIGHTLY UNBALANCED (Class 0:Class1=65:35) as it can be seen from their distribution. Since the dataset has MILD IMBALANCE (since minority class distribution is betwwen 20-40%) only, it may not lead to significant performance degradation.Hence we need not perform any method such as sampling before splitting to train and test set.

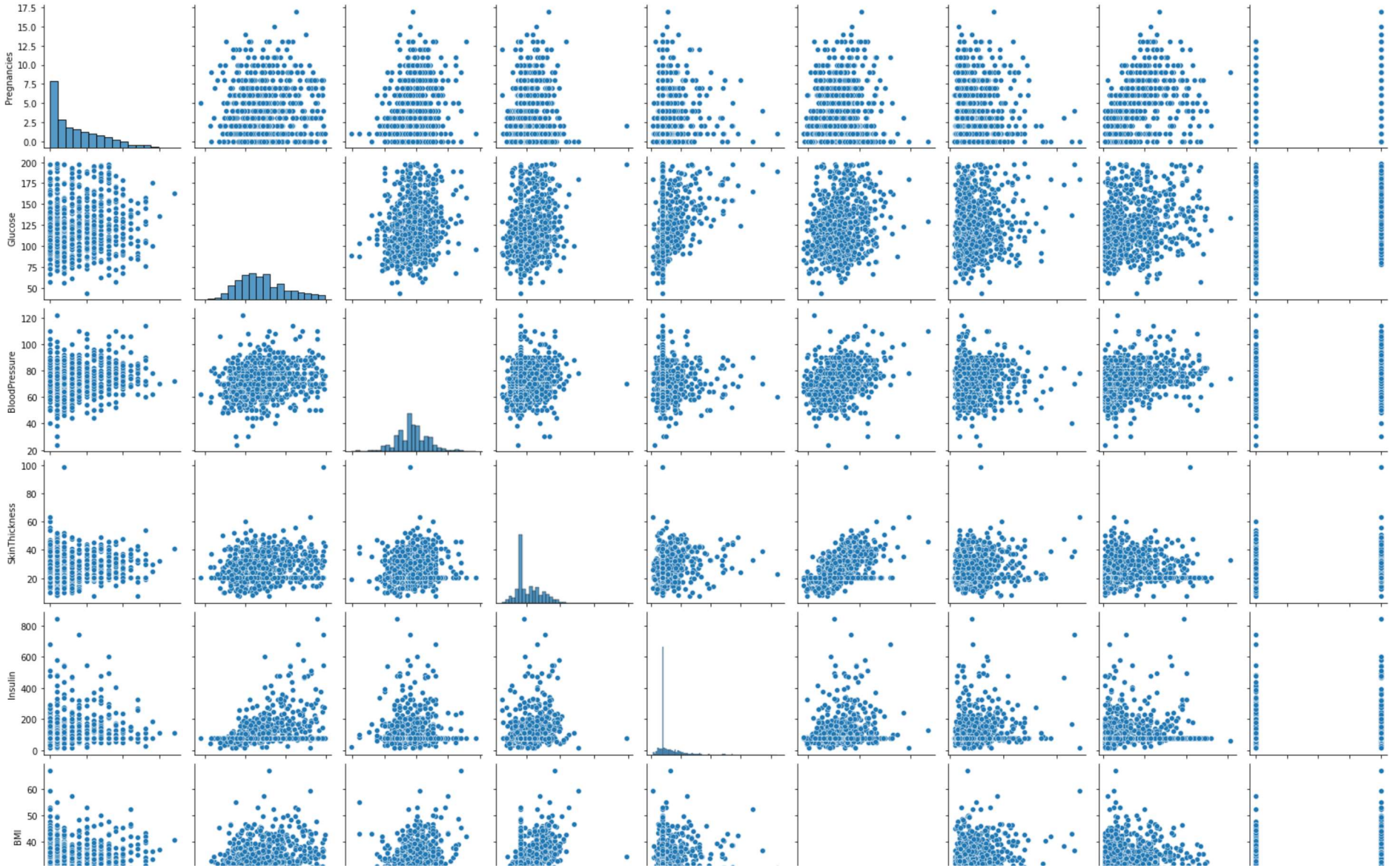
But at the time of validation of the models, we should use parameters like AUC(Area under ROC Curve) when dealing with imbalanced classes, and relying on Accuracy rate alone won't be a great idea eventhough Degree of data imabalnce is only MILD.

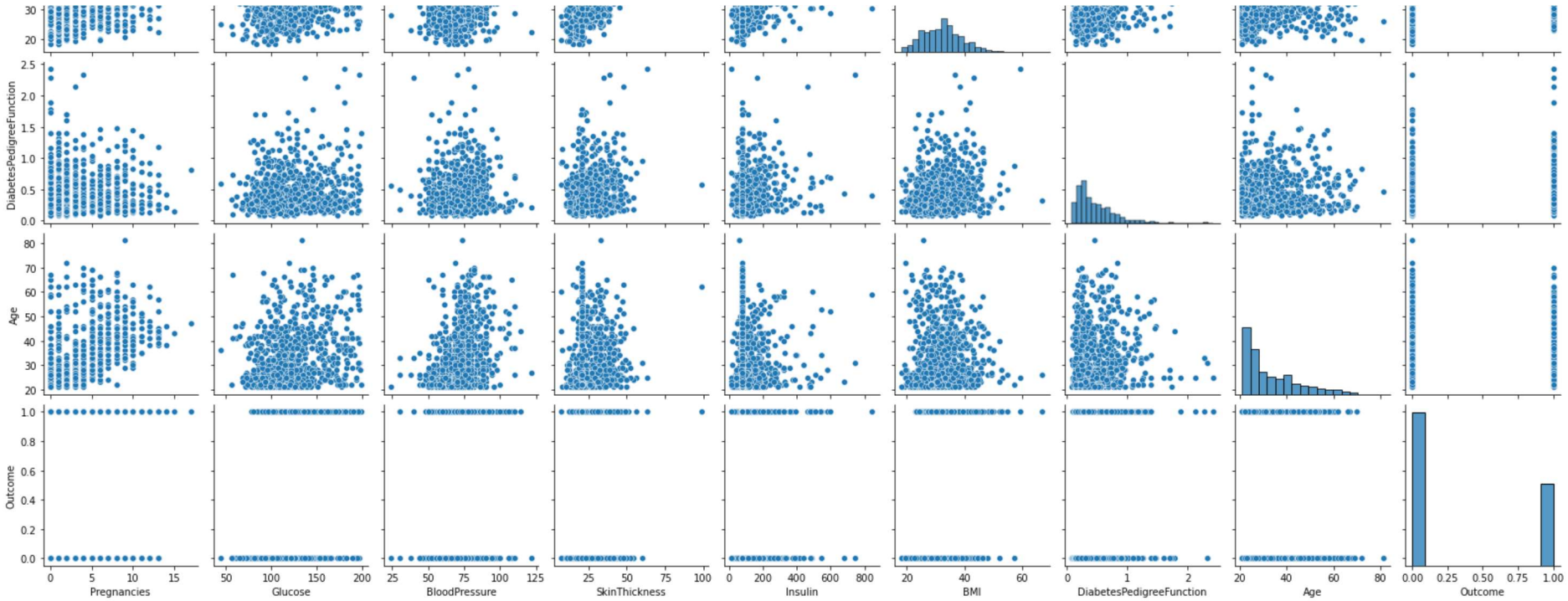
2) Create scatter charts between the pair of variables to understand the

relationships. Describe your findings.

In [18]:
```
# Using pairplot function, scatter plots between different variables plotted
sns.pairplot(data_frame)
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x1e797687790>





By analysing the scatter plot between the input variables we can take into account the relation between BMI & SkinThickness, Pregnancy and Age , Glucose and Insulin which shows a slight chance of positive correlation. But NO two variables among feature variables show strong multicolinearity, so problem of overfitting has less chance to exist.

3) Perform correlation analysis. Visually explore it using a heat map.

In [19]:
```
# Find the correlation matrix between different variables
data_frame.corr()
```

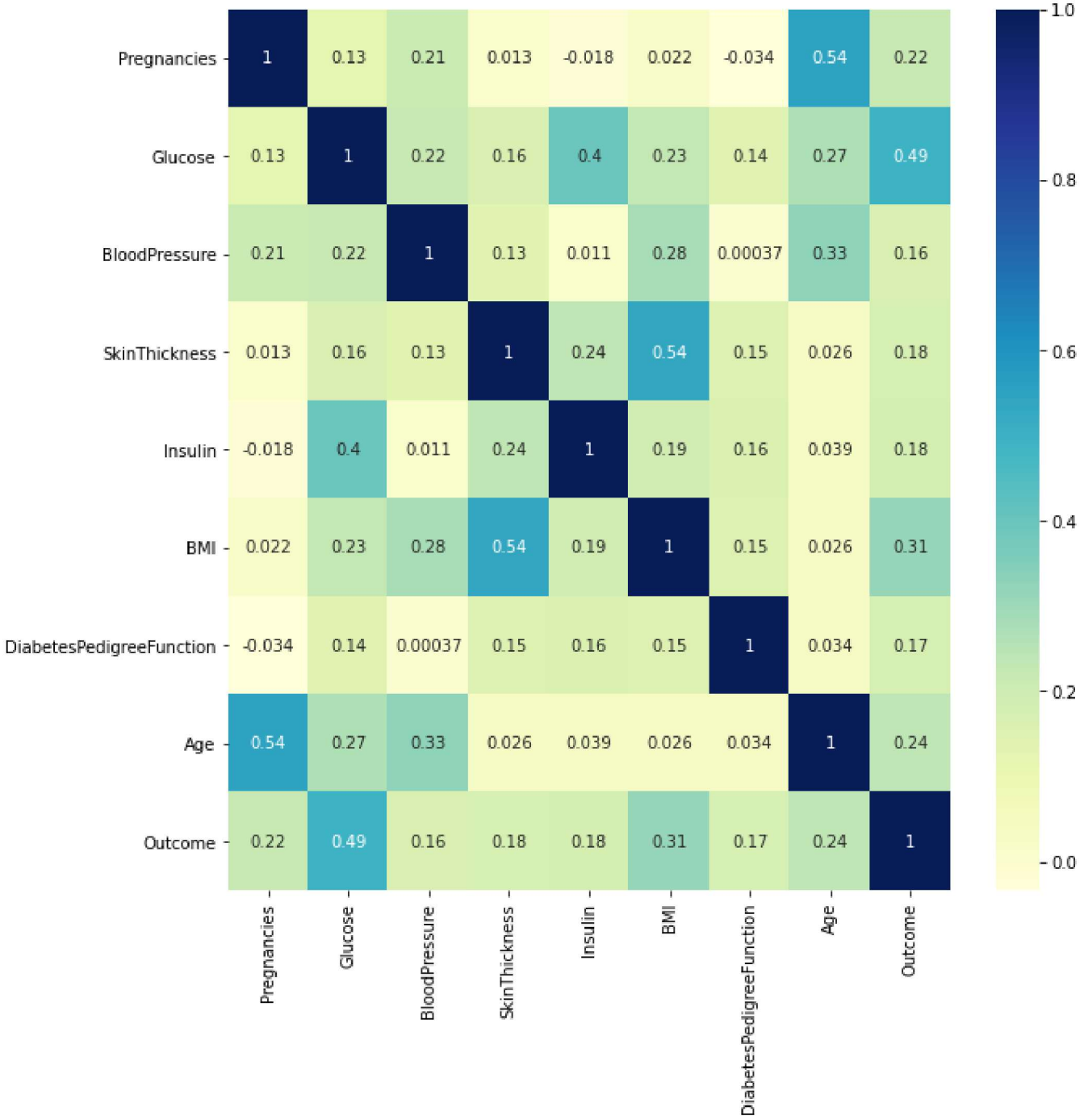
Out[19]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.127964	0.208984	0.013376	-0.018082	0.021546	-0.033523	0.544341	0.221898
Glucose	0.127964	1.000000	0.219666	0.160766	0.396597	0.231478	0.137106	0.266600	0.492908
BloodPressure	0.208984	0.219666	1.000000	0.134155	0.010926	0.281231	0.000371	0.326740	0.162986
SkinThickness	0.013376	0.160766	0.134155	1.000000	0.240361	0.535703	0.154961	0.026423	0.175026

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Insulin	-0.018082	0.396597	0.010926	0.240361	1.000000	0.189856	0.157806	0.038652	0.179185
BMI	0.021546	0.231478	0.281231	0.535703	0.189856	1.000000	0.153508	0.025748	0.312254
DiabetesPedigreeFunction	-0.033523	0.137106	0.000371	0.154961	0.157806	0.153508	1.000000	0.033561	0.173844
Age	0.544341	0.266600	0.326740	0.026423	0.038652	0.025748	0.033561	1.000000	0.238356
Outcome	0.221898	0.492908	0.162986	0.175026	0.179185	0.312254	0.173844	0.238356	1.000000

In [20]:
```
# plot Heatmap of Correlation between different variables
plt.subplots(figsize=(10,10))
sns.heatmap(data_frame.corr(),annot=True,cmap="YlGnBu")
```

Out[20]: <AxesSubplot:>


Pregnancies 1 0.13 0.21 0.013 -0.018 0.022 -0.034 0.54 0.22
Glucose 0.13 1 0.22 0.16 0.4 0.23 0.14 0.27 0.49
BloodPressure 0.21 0.22 1 0.13 0.011 0.28 0.00037 0.33 0.16
SkinThickness 0.013 0.16 0.13 1 0.24 0.54 0.15 0.026 0.18
Insulin -0.018 0.4 0.011 0.24 1 0.19 0.16 0.039 0.18
BMI 0.022 0.23 0.28 0.54 0.19 1 0.15 0.026 0.31
DiabetesPedigreeFunction -0.034 0.14 0.00037 0.15 0.16 0.15 1 0.034 0.17
Age 0.54 0.27 0.33 0.026 0.039 0.026 0.034 1 0.24
Outcome 0.22 0.49 0.16 0.18 0.18 0.31 0.17 0.24 1
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0

- On the the output variable 'Outcome', both BMI and Glucose have an effect on it.
- Positive correlation exist between input variables Age & Pregnancies , SkinThickness & BMI, Glucose & Insulin

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.

Since both Input and output variables are present in the dataset, this comes under Supervised Learning Techniques .The common machine learning tasks in supervised learning includes Classification and Regression.

Here the data contains mostly numerical data. So among Regression model, we can use Logistic Regression. Logistic regression can be used to predict binary outputs.

The output variable Outcome has values 1 or 0, ie it indicates the classification between whether the patient is Diabetic or not. So we can apply and check for the performance of classification models such as Random Forest (Ensemble Learning), Naive Bayes Classifier, SVM Classifier and Decision Tree Model.

Selection of the best model is made on the basis of the model's performance on the testing set. To validate model I will be using Accuracy score ,Confusion matrix, Recall, Preision, Specificity, F1 Score, ROC Curve and AUC.

- Accuracy Score indicates how much correct predictions are made by model over all kinds predictions made. Accuracy Score= (TP +TN)/(TP+TN+FP+FN)
- Sensitivity or Recall is the proportion of patients that were identified correctly to have diabetics (i.e. True Positive) upon the total number of patients who actually have the diabetics. Sensitivity or Recall= TP / (TP+FN)
- Specificity is the proportion of patients that were identified correctly to not have the diabetics (i.e. True Negative) upon the total number of patients who do not have the diabetics. Specificity= TN / (TN+FP)
- Precision is a measure that tells us what proportion of patients that we diagnosed as having diabetics, actually had diabetics. Precision= TP / (TP+FP)
- F1 score is a weighted harmonic mean of precision and recall. F1 score= (2 *Precision* Recall)/(Precision + Recall)

Specificity, Sensitivity(Recall), F1 Score, Precision can be obtained from the function classification_report() from sklearn

- Specificity gives us the True Negative Rate & (1—Specificity) gives us the False Positive Rate. So the sensitivity can be called as the "True Positive Rate" and (1—Specificity) can be called as the "False Positive Rate".
- ROC is a plot with Sensitivity Vs (1—Specificity) or in simple words ROC is between "True Positive Rate" and "False Positive Rate". ROC (Receiver Operating Characteristic) Curve tells us about how good the model can distinguish between two things (e.g If a patient has diabetics or not). Better models can accurately distinguish between the two.
- AUC (Area Under ROC Curve) indicates how well the probabilities from the positive classes(Class 1 or Diabetic patients) are separated from the negative classes (Class 0 or Non-Diabetic).

2) Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

In [21]:
```
# from sklearn library import necessary files for splitting data into train and test
from sklearn.model_selection import train_test_split
```

In [22]:
```
# features represents the input variables and target represents the output variable which should be given as arguments to train_te
#train_test_split() function Splits arrays into random train and test subsets, so make faetures and targets should be converted in
features =data_frame.iloc[:,0:8].values
target=data_frame.iloc[:,8].values
```

In [23]:
```
# Splitting data into train and test sets
x_train,x_test,y_train,y_test=train_test_split(features, target, train_size = .8)

# Understsnding the size of train and test data
print('Train set of features: ', x_train.shape)
print('Test set of features: ', x_test.shape)
print('Target for train: ', y_train.shape)
print('Target for test: ', y_test.shape)
```

```
Train set of features:  (614, 8)
Test set of features:  (154, 8)
Target for train:  (614,)
Target for test:  (154,)
```

In [24]:
```
# feature scaling using StandardScaler
from sklearn.preprocessing import StandardScaler
Std_Scale=StandardScaler()
x_train_std=Std_Scale.fit_transform(x_train)# Fit and transform on train data
x_test_std=Std_Scale.transform(x_test) # Transform test data
```

Logistic Regression

Here the first Supervised Learning model I am using is Logistic Regression which is widely used to predict binary outcomes for a given set of independent variables.Here the dependent variable is discrete such as Outcome ϵ {0, 1} ie non- Diabetic or Diabetic. Since the binary dependent variable has only 2 outcomes, we can apply and check for Logistic Regression Model.

In [25]:
```
# Import necesary libraries for Logistic Regression
from sklearn.linear_model import LogisticRegression
my_model_logicReg = LogisticRegression(C=0.01)
# Learn the model on train data
my_model_logicReg.fit(x_train_std,y_train)
```

Out[25]:
```
LogisticRegression(C=0.01)
```

In [26]:
```
# Predict Using Logistic Regression Model
y_pred_logicReg=my_model_logicReg.predict(x_test_std)
```

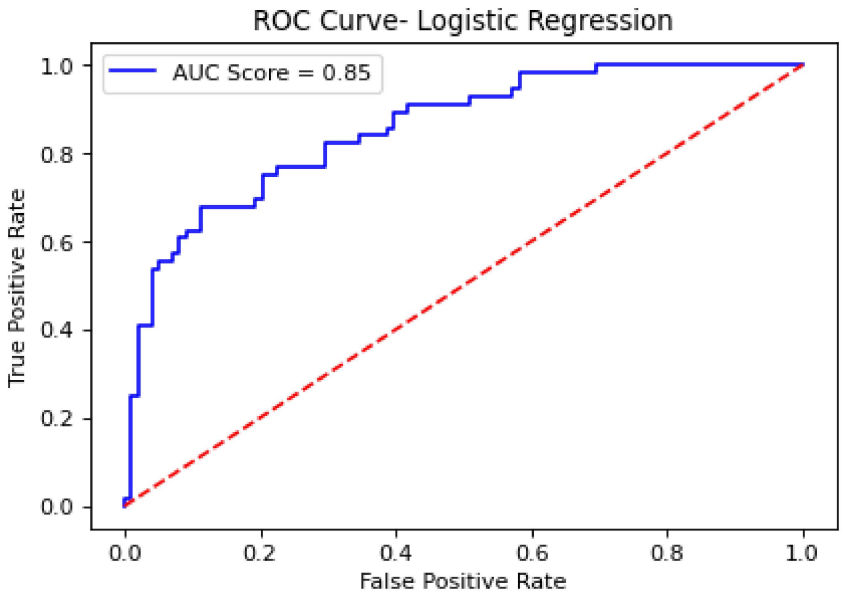
In [27]:
```
# Import the necessary libraries for Model Validation
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix,roc_auc_score,roc_curve

print("Model Validation")

# Building Confusion Matrix
matrix_logicReg=confusion_matrix(y_test,y_pred_logicReg,labels=[0,1])
print('Confusion matrix \n' ,matrix_logicReg)

# Accurcay is calculated
logicReg_acc=accuracy_score(y_test, y_pred_logicReg)
print("Accuracy Score of Logistic Regression Model::")
print(logicReg_acc)
```

```
# Classifiaction Report is generated
print("Classification Report:")
print(classification_report(y_test,y_pred_logicReg),'\n')

#Preparing ROC Curve (Receiver Operating Characteristics Curve)

# predict probabilities
logicReg_prob=my_model_logicReg.predict_proba(x_test_std)
# Find probabilities for the positive outcome
logicReg_prob1=logicReg_prob[:,1]

# calculate AUC
auc_log= roc_auc_score(y_test, logicReg_prob1)
print("AUC of Logistic Regression Model  : ",auc_log)

# calculate roc curve
fpr,tpr,thresh=roc_curve(y_test,logicReg_prob1)

# Plot ROC
plt.figure(dpi=80)
plt.title("ROC Curve- Logistic Regression")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')# calculate AUC
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%auc_log)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Model Validation
Confusion matrix
 [[91  7]
 [25 31]]
Accuracy Score of Logistic Regression Model::
0.7922077922077922
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.93      0.85        98
           1       0.82      0.55      0.66        56

    accuracy                           0.79       154
   macro avg       0.80      0.74      0.76       154
weighted avg       0.80      0.79      0.78       154
```

```
AUC of Logistic Regression Model  :  0.8549562682215744
```

Out[27]:
```
<matplotlib.legend.Legend at 0x1e79c025190>
```



SVM ALGORITHM

The first Classification model I am using is SVM algorithm that creates the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category.

In [28]:
```
# Import necesary libraries for SVM Classifier
from sklearn.svm import SVC
my_model_svm = SVC(kernel='linear',random_state=0,probability=True,C=1)
# Learn the model on train data
my_model_svm.fit(x_train_std,y_train)
```

Out[28]:
```
SVC(C=1, kernel='linear', probability=True, random_state=0)
```

In [29]:
```
# Predict Using SVM
y_pred_svm=my_model_svm.predict(x_test_std)
```

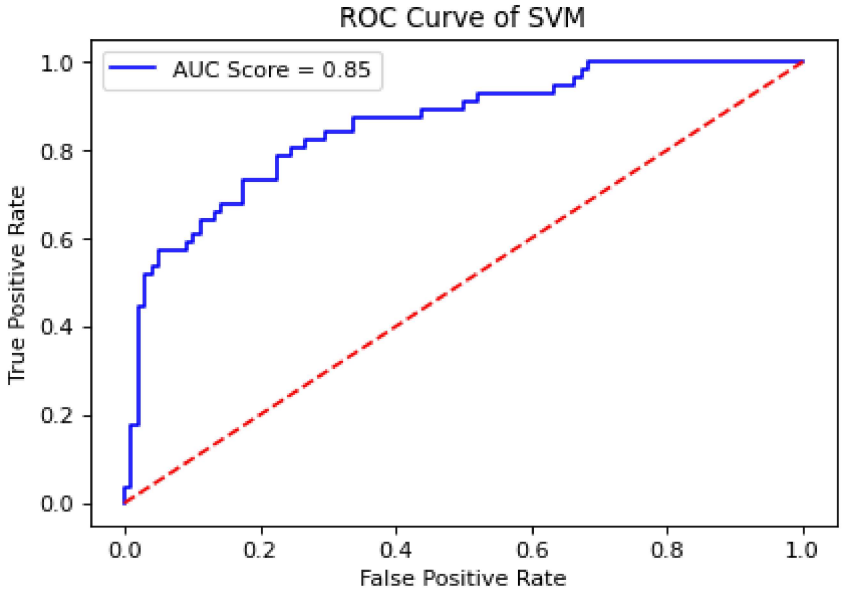
In [30]:

```
print("Model Validation of SVM ")

# Building Confusion Matrix
matrix_SVM=confusion_matrix(y_test,y_pred_svm,labels=[0,1])
print('Confusion matrix : \n',matrix_SVM)

# Accurcay is calculated
print("Accuracy Score of SVC Model with linear Kernel::")
SVM_acc=accuracy_score(y_test,y_pred_svm)
print(SVM_acc)

# Classifiaction Report is generated
print("Classification Report:")
print(classification_report(y_test,y_pred_svm))

#Preparing ROC Curve (Receiver Operating Characteristics Curve)
#Predict Probabilities
svm_prob=my_model_svm.predict_proba(x_test_std)
svm_prob1=svm_prob[:,1] # Probabilities of Positive outcomes
fpr,tpr,thresh=roc_curve(y_test,svm_prob1)

# calculate AUC
auc_svm = roc_auc_score(y_test, svm_prob1)
print("AUC of SVM Classifier Model  : ",auc_svm)

# plot ROC
plt.figure(dpi=80)
plt.title("ROC Curve of SVM")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%auc_svm)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Model Validation of SVM
Confusion matrix :
 [[88 10]
 [23 33]]
Accuracy Score of SVC Model with linear Kernel::
0.7857142857142857
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.90      0.84        98
```

```
           1       0.77      0.59      0.67        56

    accuracy                           0.79       154
   macro avg       0.78      0.74      0.75       154
weighted avg       0.78      0.79      0.78       154

AUC of SVM Classifier Model  :  0.8547740524781341
```

Out[30]: <matplotlib.legend.Legend at 0x1e79c0d7190>



KNN ALgorithm

In [31]:

```
# Import necesary libraries for KNN Algorithm
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
my_model_KNN = KNeighborsClassifier(n_neighbors = 7) # Using 7 neighbours
#Learn From Train data
my_model_KNN .fit(x_train_std,y_train)

# Predict Using KNN Model
y_pred_KNN=my_model_KNN .predict(x_test_std)
```

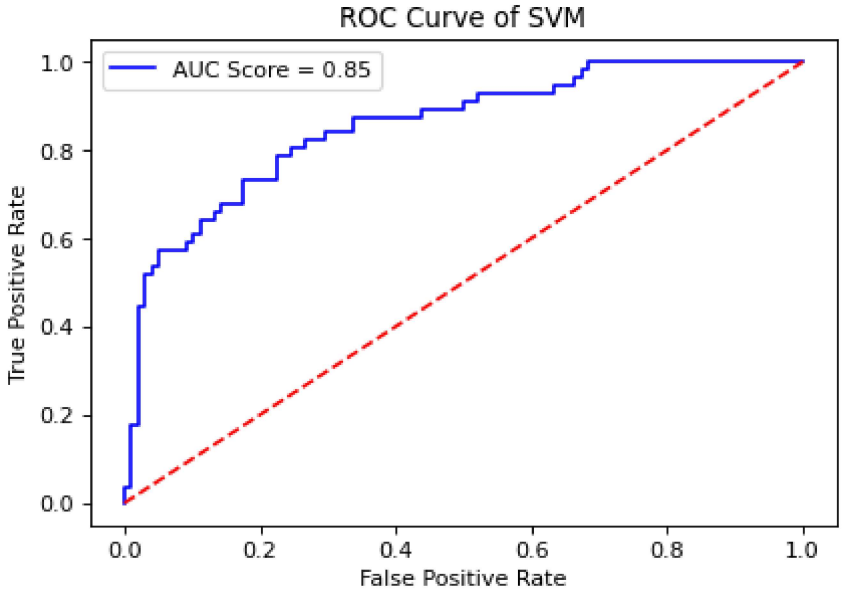
In [32]:
```
print("\n","Model Validation of KNN ","\n")

# Building Confusion Matrix
matrix_KNN=confusion_matrix(y_test,y_pred_KNN,labels=[0,1])
print('Confusion matrix',"\n",matrix_KNN)

# Accuracy Calculated
print("Accuracy Score of KNN : ")
KNN_acc=accuracy_score(y_test,y_pred_KNN)
print(KNN_acc)

# Classifiaction Report is generated
# Recall of the positive class is  “sensitivity”; recall of the negative class is “specificity”.
print("Classification Report::")
print(classification_report(y_test,y_pred_KNN),'\n')

#Preparing ROC Curve (Receiver Operating Characteristics Curve)
KNN_prob=my_model_KNN.predict_proba(x_test_std) # Predict probabilities
KNN_prob1=svm_prob[:,1] # Probabilities of Positive outcomes
fpr,tpr,thresh=roc_curve(y_test,KNN_prob1)

# calculate AUC
auc_KNN = roc_auc_score(y_test, KNN_prob1) ## **probability of the class with the greater label is given
print("AUC KNN is",auc_KNN )

#Plot ROC
plt.figure(dpi=80)
plt.title("ROC Curve of SVM")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%auc_KNN)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
 Model Validation of KNN

Confusion matrix
 [[84 14]
 [28 28]]
Accuracy Score of KNN :
0.7272727272727273
Classification Report::
              precision    recall  f1-score   support
```

```
              0       0.75      0.86      0.80        98
              1       0.67      0.50      0.57        56

    accuracy                            0.73       154
   macro avg       0.71      0.68      0.69       154
weighted avg       0.72      0.73      0.72       154
```

```
AUC KNN is 0.8547740524781341
```

Out[32]:
```
<matplotlib.legend.Legend at 0x1e79c216430>
```



Random Forest (Ensemble Learning)

Next I am using Random forests or Random decision forests which is an ensemble learning method for classification by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean/average prediction of the individual trees.

In [33]:
```
# Import necesary libraries for Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
my_model_rf = RandomForestClassifier(n_estimators=1000,random_state=0)

#Learning from train data
my_model_rf.fit(x_train_std,y_train)
```

```
# Predict Using Random Forest Model
y_pred_rf=my_model_rf.predict(x_test_std)
```

In [34]:

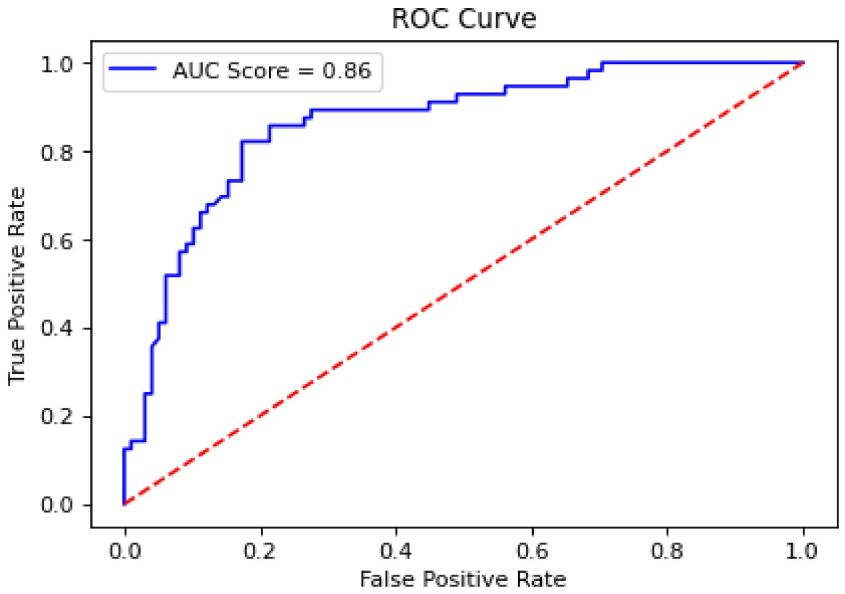
```
# Building Confusion Matrix
matrix_rf=confusion_matrix(y_test,y_pred_rf,labels=[0,1])
print('Confusion matrix : \n',matrix_rf)
#rf_acc=accuracy_score(y_pred_rf,y_test)

# Accurcay is calculated
print("Accuracy Score of Random Forest Classifier::")
RF_acc=accuracy_score(y_test,y_pred_rf)
print(RF_acc)

# Classifiaction Report is generated
print("Classification Report")
print(classification_report(y_test,y_pred_rf,target_names=['class 0','Class 1']),'\n')

#Preparing ROC Curve (Receiver Operating Characteristics Curve)
rf_prob=my_model_rf.predict_proba(x_test_std) # Predict probabilities
rf_prob1=rf_prob[:,1]# Probabilities of Positive outcome
fpr,tpr,thresh=roc_curve(y_test,rf_prob1)

# calculate AUC
auc_RF = roc_auc_score(y_test, rf_prob1)
print("AUC Random Forest is",auc_RF )

#roc_auc_rf=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%auc_RF)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Confusion matrix :
 [[87 11]
 [20 36]]
Accuracy Score of Random Forest Classifier::
0.7987012987012987
Classification Report
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

     class 0       0.81      0.89      0.85        98
     Class 1       0.77      0.64      0.70        56

    accuracy                           0.80       154
   macro avg       0.79      0.77      0.77       154
weighted avg       0.80      0.80      0.79       154
```

```
AUC Random Forest is 0.8631559766763849
```

Out[34]:
```
<matplotlib.legend.Legend at 0x1e79d3e0a60>
```



Decision tree

Next Classification technique we are using is decision tree which is a tree-like graph with nodes representing the place where we pick an attribute and ask a question; edges represent the answers the to the question; and the leaves represent the actual output or class label. They are used in non-linear decision making with simple linear decision surface.

In [35]:
```
#importing library for Descision Tree
from sklearn.tree import DecisionTreeClassifier
# you can use gini since it is binary classification otherwise use entropy
my_model_DT = DecisionTreeClassifier(criterion = 'gini', random_state = 0,
                                     max_depth = 2, min_samples_leaf = 10, min_samples_split = 20)
```

```
# Learn using the train data
my_model_DT.fit(x_train, y_train) #Decision tree doesn't require feature normalization, that's because the model only needs the ab
```

Out[35]:
```
DecisionTreeClassifier(max_depth=2, min_samples_leaf=10, min_samples_split=20,
                       random_state=0)
```

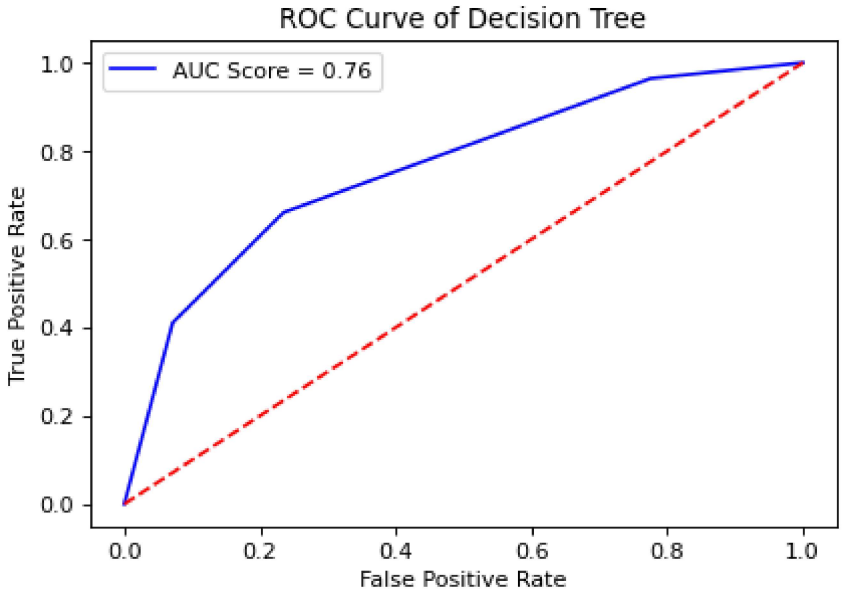
In [36]:
```
# Predict Using Decision Tree Classifier
y_pred_DT = my_model_DT.predict(x_test)
```

In [37]:
```
# Building Confusion Matrix
matrix_DT = confusion_matrix(y_test, y_pred_DT,labels=[0,1])
print('Confusion matrix : \n',matrix_DT )

# Accurcay is calculated
DT_acc = accuracy_score(y_test,y_pred_DT)
print('Accuracy of Decision Tree Classifier is ',DT_acc)

# Classifiaction Report is generated
print(classification_report(y_test,y_pred_DT))

#Preparing ROC Curve (Receiver Operating Characteristics Curve)
DT_prob=my_model_DT.predict_proba(x_test)# Predict probabilities
DT_prob1=DT_prob[:,1]# Probabilities of Positive outcome
fpr,tpr,thresh=roc_curve(y_test,DT_prob1)

# calculate AUC
auc_DT = roc_auc_score(y_test, DT_prob1)
print("AUC Deccision Tree Classifier  is",auc_DT )

#plot ROC
plt.figure(dpi=80)
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%auc_DT)
plt.title("ROC Curve of Decision Tree")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Confusion matrix :
 [[91  7]
 [33 23]]
Accuracy of Decision Tree Classifier is  0.7402597402597403
```

```
              precision    recall  f1-score   support

           0       0.73      0.93      0.82        98
           1       0.77      0.41      0.53        56

    accuracy                           0.74       154
   macro avg       0.75      0.67      0.68       154
weighted avg       0.75      0.74      0.72       154

AUC Deccision Tree Classifier  is 0.7620262390670554
```

Out[37]: <matplotlib.legend.Legend at 0x1e79d453880>



Naive Bayes Classifier

Naive Bayer Classification technique is a probabilistic machine learning model that works based on Baye's theorem.

In [38]:

```
# Importing the necessary library for Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
my_model_NB = GaussianNB()

# Learn using the Model
my_model_NB.fit(x_train, y_train)
```

```
# Predict Using Naive Bayes Classifier
y_pred_NB = my_model_NB.predict(x_test)
```

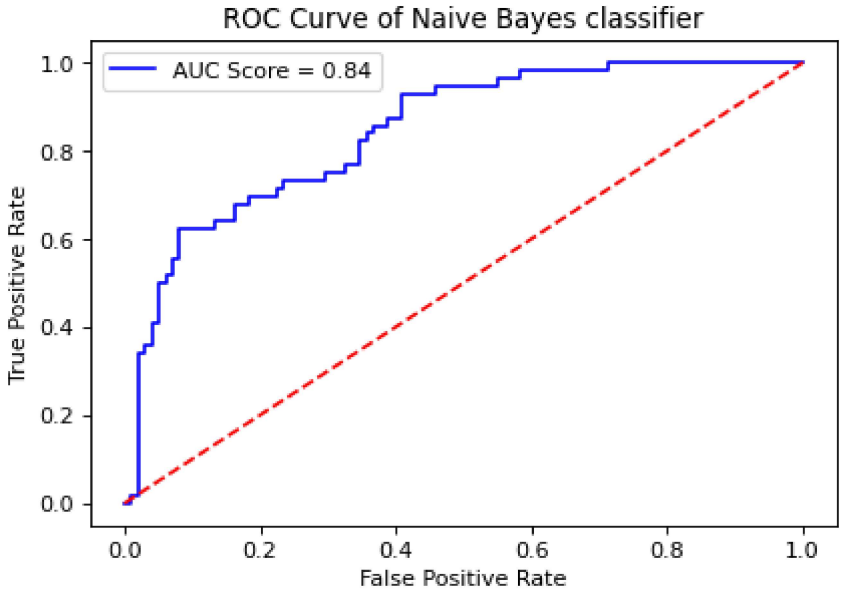
In [39]:

```
#Building Confusion Matrix
matrix_NB = confusion_matrix(y_test,y_pred_NB,labels=[0,1])
print('Confusion matrix : \n',matrix_NB )

# Accurcay is calculated
NB_acc = accuracy_score(y_test,y_pred_NB)
print('Accuracy of Decision Tree Classifier is ',NB_acc)

# Classifiaction Report is generated
print(classification_report(y_test,y_pred_NB))

#Preparing ROC Curve (Receiver Operating Characteristics Curve)
NB_prob=my_model_NB.predict_proba(x_test)# Predict probabilities
NB_prob1=NB_prob[:,1]# Probabilities of Positive outcome
fpr,tpr,thresh=roc_curve(y_test,NB_prob1)

# calculate AUC
auc_NB = roc_auc_score(y_test, NB_prob1)
print("AUC of Naive Bayes Classifier  is",auc_NB )

# Plot ROC
plt.figure(dpi=80)
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%auc_NB)
plt.title("ROC Curve of Naive Bayes classifier")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Confusion matrix :
 [[82 16]
 [20 36]]
Accuracy of Decision Tree Classifier is  0.7662337662337663
              precision    recall  f1-score   support

           0       0.80      0.84      0.82        98
           1       0.69      0.64      0.67        56

    accuracy                           0.77       154
   macro avg       0.75      0.74      0.74       154
```

```
weighted avg       0.76      0.77      0.76       154

AUC of Naive Bayes Classifier  is 0.8442055393586005
```

Out[39]: `<matplotlib.legend.Legend at 0x1e79d4c2640>`



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.

In [40]:
```
# Compare various models by their accuracy rates
My_models=['Logistic Regression','SVM','KNN','Random Forest','Decision Tree','Naive bayes']
My_models_accuracy = logicReg_acc,SVM_acc,KNN_acc,RF_acc,DT_acc,NB_acc # accuracy rates of different models we used
Accuracy_Scores=pd.DataFrame({'MODELS':My_models,'ACCURACY':My_models_accuracy})
Accuracy_Scores
```

Out[40]:

	MODELS	ACCURACY
0	Logistic Regression	0.792208

	MODELS	ACCURACY
1	SVM	0.785714
2	KNN	0.727273
3	Random Forest	0.798701
4	Decision Tree	0.740260
5	Naive bayes	0.766234

In [41]:
```
# Find the model having the highest accutracy rate
Accuracy_Scores[(Accuracy_Scores['ACCURACY']) == max(Accuracy_Scores['ACCURACY'])]
```

Out[41]:

	MODELS	ACCURACY
3	Random Forest	0.798701

- From the above analysis we can understand that maximum Accuracy rate is for Random Forest Classifier compared to other models. Since accuracy is not the only performance metric we rely upon, let us consider other parameters too for finding the appropriate model for Prediction.

In [45]:
```
# Compare various models by their AUC (Area Under ROC Curve)
My_models=['Logistic Regression','SVM','KNN','Random Forest','Decision Tree','Naive bayes']
My_models_auc = auc_log,auc_svm,auc_KNN,auc_RF,auc_DT,auc_NB
AUC_Scores=pd.DataFrame({'MODELS':My_models,'AUC':My_models_auc})
AUC_Scores
```

Out[45]:

	MODELS	AUC
0	Logistic Regression	0.854956
1	SVM	0.854774
2	KNN	0.854774
3	Random Forest	0.863156
4	Decision Tree	0.762026
5	Naive bayes	0.844206

In [46]:
```
# Find the model having the highest AUC score
AUC_Scores[(AUC_Scores['AUC']) == max(AUC_Scores['AUC'])]
```

Out[46]:

	MODELS	AUC
3	Random Forest	0.863156

- While analysing the AUC (Area Under ROC Curve), we can find that Random Forest Classifier has highest AUC score.

The performance metrics of Random Forest Classifier is once again genereted to analyze the values

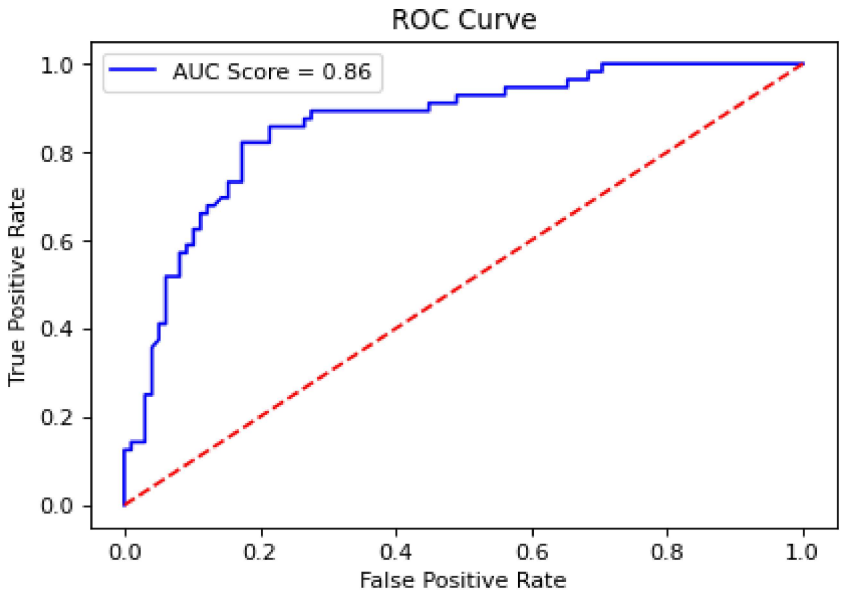
In [47]:
```
#  Confusion Matrix of Random Forest Classifier
matrix_rf=confusion_matrix(y_test,y_pred_rf,labels=[0,1])
print('Confusion matrix : \n',matrix_rf)

# Accurcay of Random Forest Classifier
print("Accuracy Score of Random Forest Classifier::")
RF_acc=accuracy_score(y_test,y_pred_rf)
print(RF_acc)

# Classifiaction Report of Random Forest Classifier
print("Classification Report")
print(classification_report(y_test,y_pred_rf,target_names=['class 0','Class 1']),'\n')

#Preparing ROC Curve (Receiver Operating Characteristics Curve) of  Random Forest Classifier
rf_prob=my_model_rf.predict_proba(x_test_std) # Predict probabilities
rf_prob1=rf_prob[:,1]# Probabilities of Positive outcomes
fpr,tpr,thresh=roc_curve(y_test,rf_prob1)

# calculate AUC of Random Forest Classifier
auc_RF = roc_auc_score(y_test, rf_prob1)
print("AUC Random Forest is",auc_RF )

# Plot ROC of Random Forest Classifier
plt.figure(dpi=80)
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%auc_RF)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Confusion matrix :
 [[87 11]
 [20 36]]
Accuracy Score of Random Forest Classifier::
0.7987012987012987
Classification Report
              precision    recall  f1-score   support

     class 0       0.81      0.89      0.85        98
     Class 1       0.77      0.64      0.70        56

    accuracy                           0.80       154
   macro avg       0.79      0.77      0.77       154
weighted avg       0.80      0.80      0.79       154
```

```
AUC Random Forest is 0.8631559766763849
```

Out[47]: <matplotlib.legend.Legend at 0x1e79e57a760>



ANALYSIS OF CLASSIFICATION REPORT OF RANDOM FOREST CLASSIFIER MODEL TO CHECK WHETHER IT IS THE MOST APPROPRIATE MODEL

From the cofusion matrix we can understand that :

- TN (True Negative) = 87
- FP (False Positive) = 11
- FN(False Negative) =20
- TP(True Positive)= 36
- Accuracy Score= (TP +TN)/(TP+TN+FP+FN) = .798 ie 79.8% correct predictions are made by the Random Forest model over all kinds predictions made.

From Classification Report we can underatand that:

- Sensitivity (Recall of Positive Class) = TP / (TP+FN) = 36/(36 +20) = 0.64 ie 64 % of patients were correctly identified as DIABETIC upon the total number of patients who are actually DIABETIC.
- Specificity (Recall of Negative Class)= TN / (TN+FP) = 87 /(87+ 11) = 0.89 ie 89 % of patients were correctly identified as NON- DIABETIC upon the total number of patients who are actually NON- DIABETIC.
    - Precision (of Class 1)= TP / (TP+FP)= 0.77 ie 77 % of patients that we predicted as having diabetics, actually had diabetics.
    - Precision (of Class 0)= TN / (TN+FN)= 0.81 ie 81 % of patients that we predicted as NON- DIABETIC, actually are NON- DIABETIC .
    - F1 score (of Class 1)= (2 *Precision* Recall)/(Precision + Recall) = 0.70 , which is weighted harmonic mean of precision and recall.
    - F1 score (of Class 0)= 0.85
    - ROC(Receiver Operating Characteristics Curve) is a plotted for Random Classifier with Sensitivity Vs (1—Specificity) which represents how good the model can distinguish between Diabetic and Non - Diabetic Class
    - AUC (Area Under ROC Curve) Score = 0.86 indicates how well the probabilities from the positive classes(Diabetic Class) are separated from the negative classes (Non-Diabetic Class).
- #### By comapring all above performance parameters of Random Classifier Model with that of other models, we can conclude that it is the most appropriate model for predicting whether the patients in the dataset have diabetes or not.