In [1]:
```
#importing libraries to retrieve data and read csv and to perform EDA tasks
import pandas as pd
import numpy as np
#importing for visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df=pd.read_csv("C://Users//sumas//Documents//Data science//Capstone//project//Healthcare
```

In [3]:
```
df.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
```

no nulls in the dataset

In [4]:
```
df.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]:
```
df.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.00000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.47187
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.33132
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.07800
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.24375
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.37250
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.62625
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.42000

There is 768 observations of 9 variable. Independent variables are Pregnencies , Glucose, BloodPressure, Insulin, BMI, DiabetesPedigree Function and Age. Outcome is dependent Variable. Average Age of Patients is 33.24 with minimum being 21 yearsand maximum 81 years. Avg. value of independent variables are Preg = 3.845052,Glucose = 120.894531, BP = 69.105469, ST=20.536458, Insulin = 79.799479; BMI = 31.992578 DPF = 0.471876 and Age = 33 . Variation in variable can be easily observed from std in the above table.

In [6]:
```
df.columns
```

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

In [7]:
```
df["SkinThickness"].unique()
```

Out[7]:
```
array([35, 29,  0, 23, 32, 45, 19, 47, 38, 30, 41, 33, 26, 15, 36, 11, 31,
       37, 42, 25, 18, 24, 39, 27, 21, 34, 10, 60, 13, 20, 22, 28, 54, 40,
       51, 56, 14, 17, 50, 44, 12, 46, 16,  7, 52, 43, 48,  8, 49, 63, 99],
      dtype=int64)
```

In [8]:
```
df["Insulin"].unique()
```

Out[8]:
```
array([  0,  94, 168,  88, 543, 846, 175, 230,  83,  96, 235, 146, 115,
       140, 110, 245,  54, 192, 207,  70, 240,  82,  36,  23, 300, 342,
       304, 142, 128,  38, 100,  90, 270,  71, 125, 176,  48,  64, 228,
        76, 220,  40, 152,  18, 135, 495,  37,  51,  99, 145, 225,  49,
        50,  92, 325,  63, 284, 119, 204, 155, 485,  53, 114, 105, 285,
       156,  78, 130,  55,  58, 160, 210, 318,  44, 190, 280,  87, 271,
       129, 120, 478,  56,  32, 744, 370,  45, 194, 680, 402, 258, 375,
       150,  67,  57, 116, 278, 122, 545,  75,  74, 182, 360, 215, 184,
        42, 132, 148, 180, 205,  85, 231,  29,  68,  52, 255, 171,  73,
       108,  43, 167, 249, 293,  66, 465,  89, 158,  84,  72,  59,  81,
       196, 415, 275, 165, 579, 310,  61, 474, 170, 277,  60,  14,  95,
       237, 191, 328, 250, 480, 265, 193,  79,  86, 326, 188, 106,  65,
       166, 274,  77, 126, 330, 600, 185,  25,  41, 272, 321, 144,  15,
       183,  91,  46, 440, 159, 540, 200, 335, 387,  22, 291, 392, 178,
       127, 510,  16, 112], dtype=int64)
```

In [9]:
```
df["Glucose"].unique()
```

Out[9]:
```
array([148,  85, 183,  89, 137, 116,  78, 115, 197, 125, 110, 168, 139,
       189, 166, 100, 118, 107, 103, 126,  99, 196, 119, 143, 147,  97,
       145, 117, 109, 158,  88,  92, 122, 138, 102,  90, 111, 180, 133,
       106, 171, 159, 146,  71, 105, 101, 176, 150,  73, 187,  84,  44,
       141, 114,  95, 129,  79,   0,  62, 131, 112, 113,  74,  83, 136,
        80, 123,  81, 134, 142, 144,  93, 163, 151,  96, 155,  76, 160,
       124, 162, 132, 120, 173, 170, 128, 108, 154,  57, 156, 153, 188,
       152, 104,  87,  75, 179, 130, 194, 181, 135, 184, 140, 177, 164,
        91, 165,  86, 193, 191, 161, 167,  77, 182, 157, 178,  61,  98,
       127,  82,  72, 172,  94, 175, 195,  68, 186, 198, 121,  67, 174,
       199,  56, 169, 149,  65, 190], dtype=int64)
```

In [10]:
```
df["BloodPressure"].unique()
```

Out[10]:
```
array([ 72,  66,  64,  40,  74,  50,   0,  70,  96,  92,  80,  60,  84,
        30,  88,  90,  94,  76,  82,  75,  58,  78,  68, 110,  56,  62,
        85,  86,  48,  44,  65, 108,  55, 122,  54,  52,  98, 104,  95,
        46, 102, 100,  61,  24,  38, 106, 114], dtype=int64)
```

In [11]:
```
df["BMI"].unique()
```

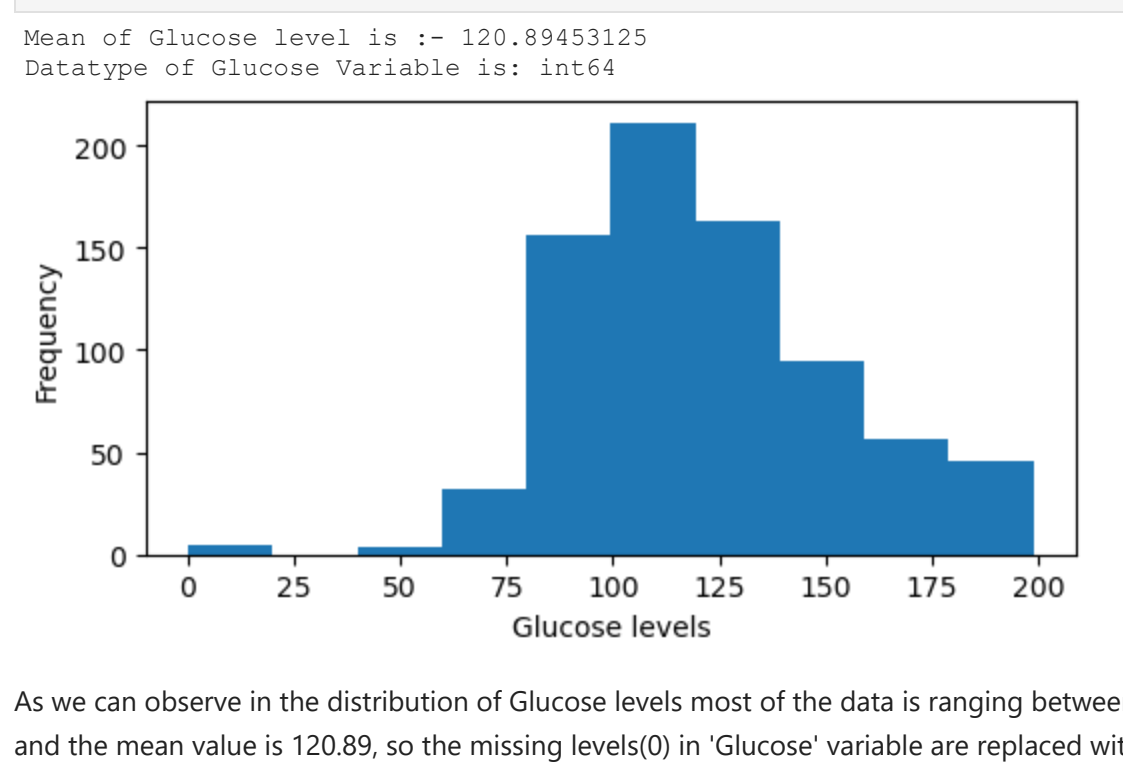
Out[11]:
```
array([33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31. , 35.3, 30.5,  0. , 37.6,
       38. , 27.1, 30.1, 25.8, 30. , 45.8, 29.6, 43.3, 34.6, 39.3, 35.4,
       39.8, 29. , 36.6, 31.1, 39.4, 23.2, 22.2, 34.1, 36. , 31.6, 24.8,
       19.9, 27.6, 24. , 33.2, 32.9, 38.2, 37.1, 34. , 40.2, 22.7, 45.4,
       27.4, 42. , 29.7, 28. , 39.1, 19.4, 24.2, 24.4, 33.7, 34.7, 23. ,
       37.7, 46.8, 40.5, 41.5, 25. , 25.4, 32.8, 32.5, 42.7, 19.6, 28.9,
       28.6, 43.4, 35.1, 32. , 24.7, 32.6, 43.2, 22.4, 29.3, 24.6, 48.8,
       32.4, 38.5, 26.5, 19.1, 46.7, 23.8, 33.9, 20.4, 28.7, 49.7, 39. ,
       26.1, 22.5, 39.6, 29.5, 34.3, 37.4, 33.3, 31.2, 28.2, 53.2, 34.2,
       26.8, 55. , 42.9, 34.5, 27.9, 38.3, 21.1, 33.8, 30.8, 36.9, 39.5,
       27.3, 21.9, 40.6, 47.9, 50. , 25.2, 40.9, 37.2, 44.2, 29.9, 31.9,
       28.4, 43.5, 32.7, 67.1, 45. , 34.9, 27.7, 35.9, 22.6, 33.1, 30.4,
       52.3, 24.3, 22.9, 34.8, 30.9, 40.1, 23.9, 37.5, 35.5, 42.8, 42.6,
       41.8, 35.8, 37.8, 28.8, 23.6, 35.7, 36.7, 45.2, 44. , 46.2, 35. ,
       43.6, 44.1, 18.4, 29.2, 25.9, 32.1, 36.3, 40. , 25.1, 27.5, 45.6,
       27.8, 24.9, 25.3, 37.9, 27. , 26. , 38.7, 20.8, 36.1, 30.7, 32.3,
       52.9, 21. , 39.7, 25.5, 26.2, 19.3, 38.1, 23.5, 45.5, 23.1, 39.9,
       36.8, 21.8, 41. , 42.2, 34.4, 27.2, 36.5, 29.8, 39.2, 38.4, 36.2,
       48.3, 20. , 22.3, 45.7, 23.7, 22.1, 42.1, 42.4, 18.2, 26.4, 45.3,
       37. , 24.5, 32.2, 59.4, 21.2, 26.7, 30.2, 46.1, 41.3, 38.8, 35.6,
       42.3, 40.7, 46.5, 33.5, 37.3, 30.3, 26.3, 21.7, 36.4, 28.5, 26.9,
       38.6, 31.3, 19.5, 20.1, 40.8, 23.4, 28.3, 38.9, 57.3, 35.2, 49.6,
       44.6, 24.1, 44.5, 41.2, 49.3, 46.3])
```

Glucose, BloodPressure, SkinThickness, Insulin, BMI have zero values which does not make any sense and indicates missing values

Visually explore variables using histograms and treat the missing values accordingly

In [12]:
```
### Glucose levels
plt.figure(figsize=(6,3),dpi=100)
plt.xlabel('Glucose levels')
df['Glucose'].plot.hist()
print("Mean of Glucose level is :-", df['Glucose'].mean())
print("Datatype of Glucose Variable is:",df['Glucose'].dtypes)
```

```
Mean of Glucose level is :- 120.89453125
Datatype of Glucose Variable is: int64
```



As we can observe in the distribution of Glucose levels most of the data is ranging between 100 and 120 and the mean value is 120.89, so the missing levels(0) in 'Glucose' variable are replaced with mean of Glucose level.

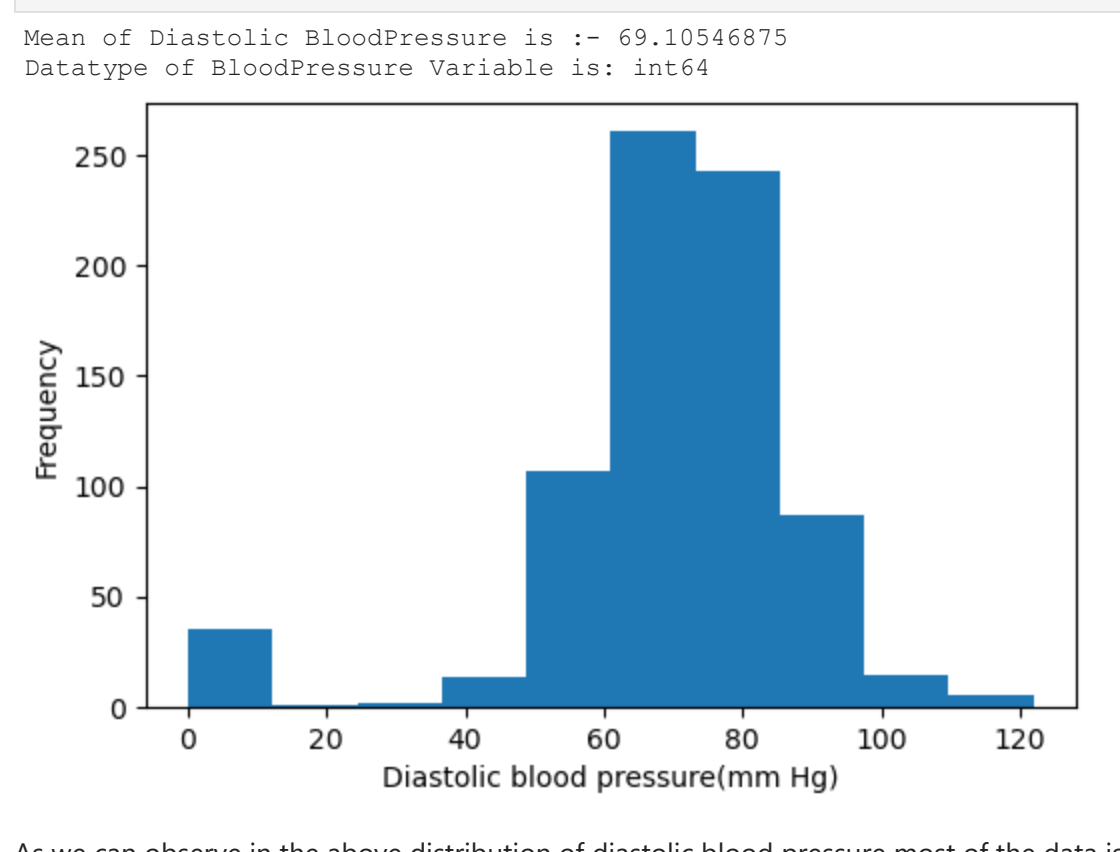
In [13]:
```
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
```

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

Out[14]:
```
99.0     17
100.0    17
111.0    14
106.0    14
125.0    14
         ..
61.0      1
190.0     1
169.0     1
172.0     1
56.0      1
Name: Glucose, Length: 136, dtype: int64
```

In [15]:
```
### Blood pressure
plt.figure(figsize=(4,4),dpi=100)
plt.xlabel('Diastolic BloodPressure(mm Hg)')
df['BloodPressure'].plot.hist()
print("Mean of Diastolic BloodPressure is :-", df['BloodPressure'].mean())
print("Datatype of BloodPressure Variable is:",df['BloodPressure'].dtypes)
```

```
Mean of Diastolic BloodPressure is :- 69.10546875
Datatype of BloodPressure Variable is: int64
```



As we can observe in the above distribution of diastolic blood pressure most of the data is ranging between 60 and 80 and the mean value is 69.10546875, so the missing levels(0) in 'BloodPressure' variable are replaced with mean of Diastolic BloodPressure.

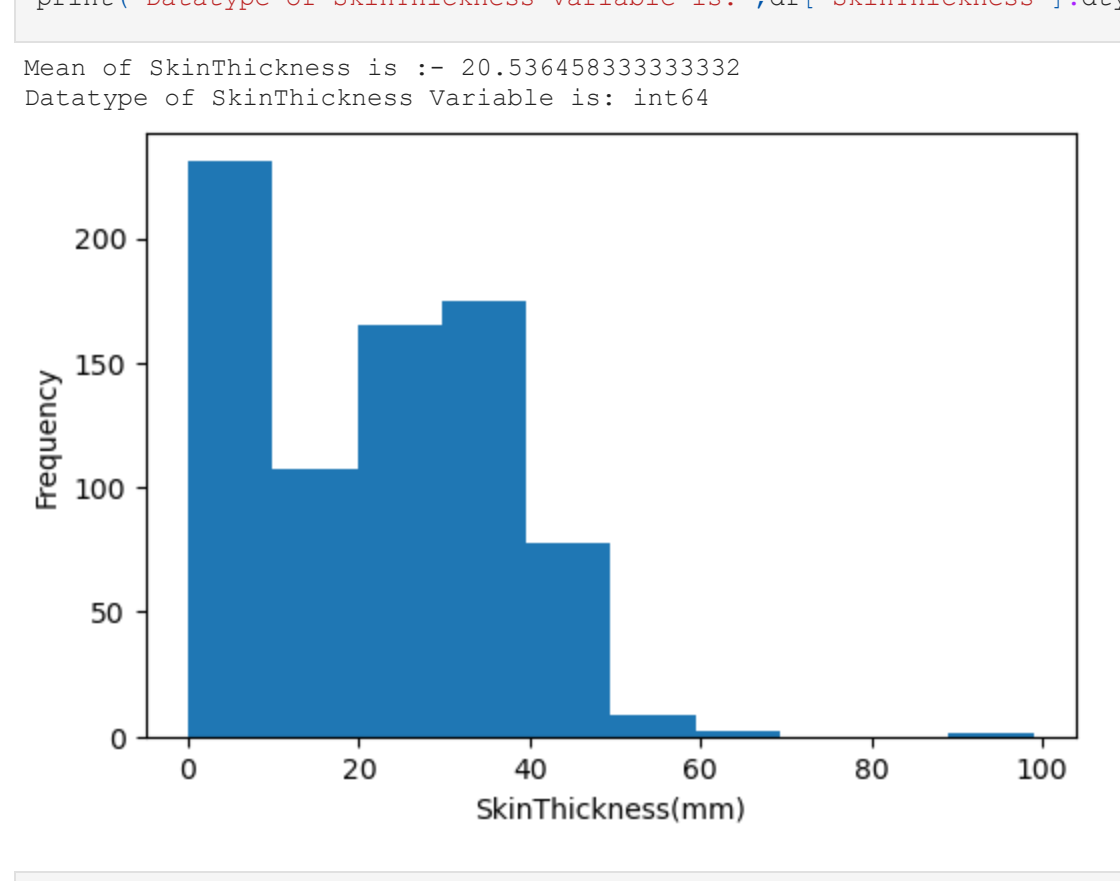
In [16]:
```
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())
```

In [17]:
```
df['BloodPressure'].value_counts().head()
```

Out[17]:
```
70.0    57
74.0    52
78.0    45
68.0    45
72.0    44
Name: BloodPressure, dtype: int64
```

In [18]:
```
### Skin Thickness variable
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel('SkinThickness(mm)')
df['SkinThickness'].plot.hist()
print("Mean of SkinThickness is :-", df['SkinThickness'].mean())
print("Datatype of SkinThickness Variable is:",df['SkinThickness'].dtypes)
```

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



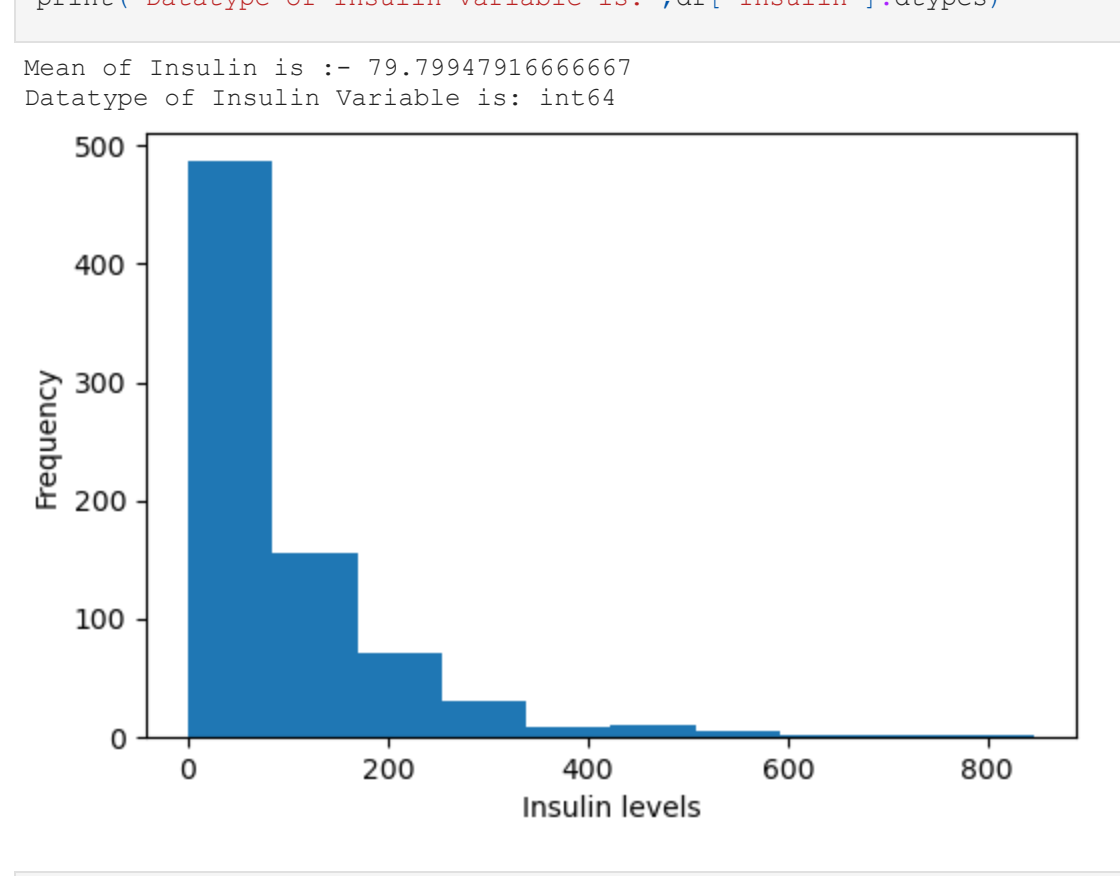
In [19]:
```
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].mean())
```

In [20]:
```
df['SkinThickness'].value_counts().head()
```

Out[20]:
```
20.536458    227
32.000000     31
30.000000     27
27.000000     23
23.000000     22
Name: SkinThickness, dtype: int64
```

In [21]:
```
### Insulin levels
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel('Insulin levels')
df['Insulin'].plot.hist()
print("Mean of Insulin is :-", df['Insulin'].mean())
print("Datatype of Insulin Variable is:",df['Insulin'].dtypes)
```

```
Mean of Insulin is :- 79.79947916666667
Datatype of Insulin Variable is: int64
```



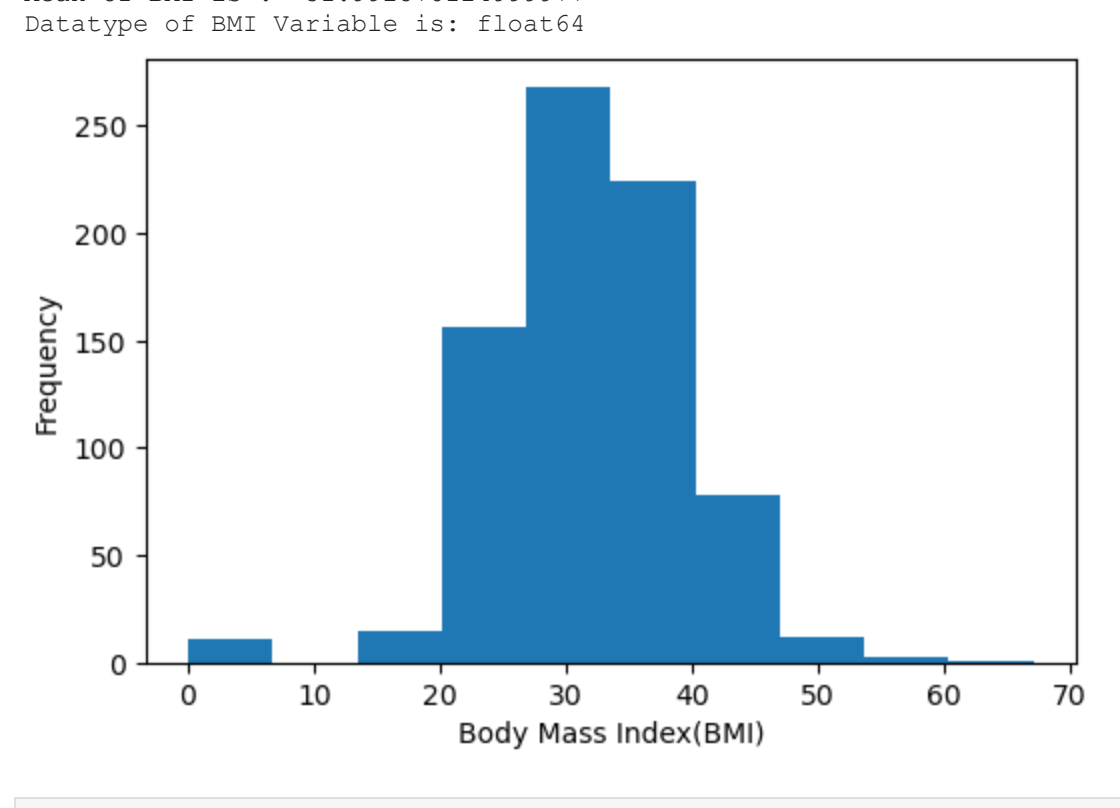
In [22]:
```
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].mean())
```

In [23]:
```
df['Insulin'].value_counts().head()
```

Out[23]:
```
79.799479     374
105.000000     11
130.000000      9
140.000000      9
120.000000      8
Name: Insulin, dtype: int64
```

In [24]:
```
### Body Mass Index
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel('Body Mass Index(BMI)')
df['BMI'].plot.hist()
print("Mean of BMI is :-", df['BMI'].mean())
print("Datatype of BMI Variable is:",df['BMI'].dtypes)
```

```
Mean of BMI is :- 31.992578124999977
Datatype of BMI Variable is: float64
```



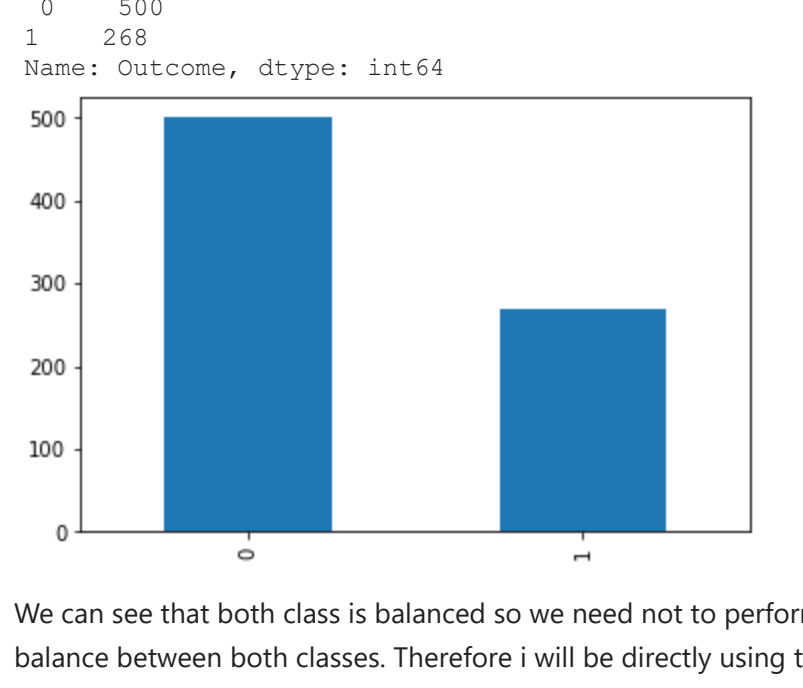
In [25]:
```
df['BMI']=df['BMI'].replace(0,df['BMI'].mean())
```

In [26]:
```
df['BMI'].value_counts().head()
```

Out[26]:
```
32.000000    13
31.600000    12
31.200000    12
31.992578    11
32.400000    10
Name: BMI, dtype: int64
```

In [27]:
```
#Check the balance of the data by plotting the count of outcomes by their value
df['Outcome'].value_counts().plot(kind='bar')
print("Count of class is:\n",df['Outcome'].value_counts())
```

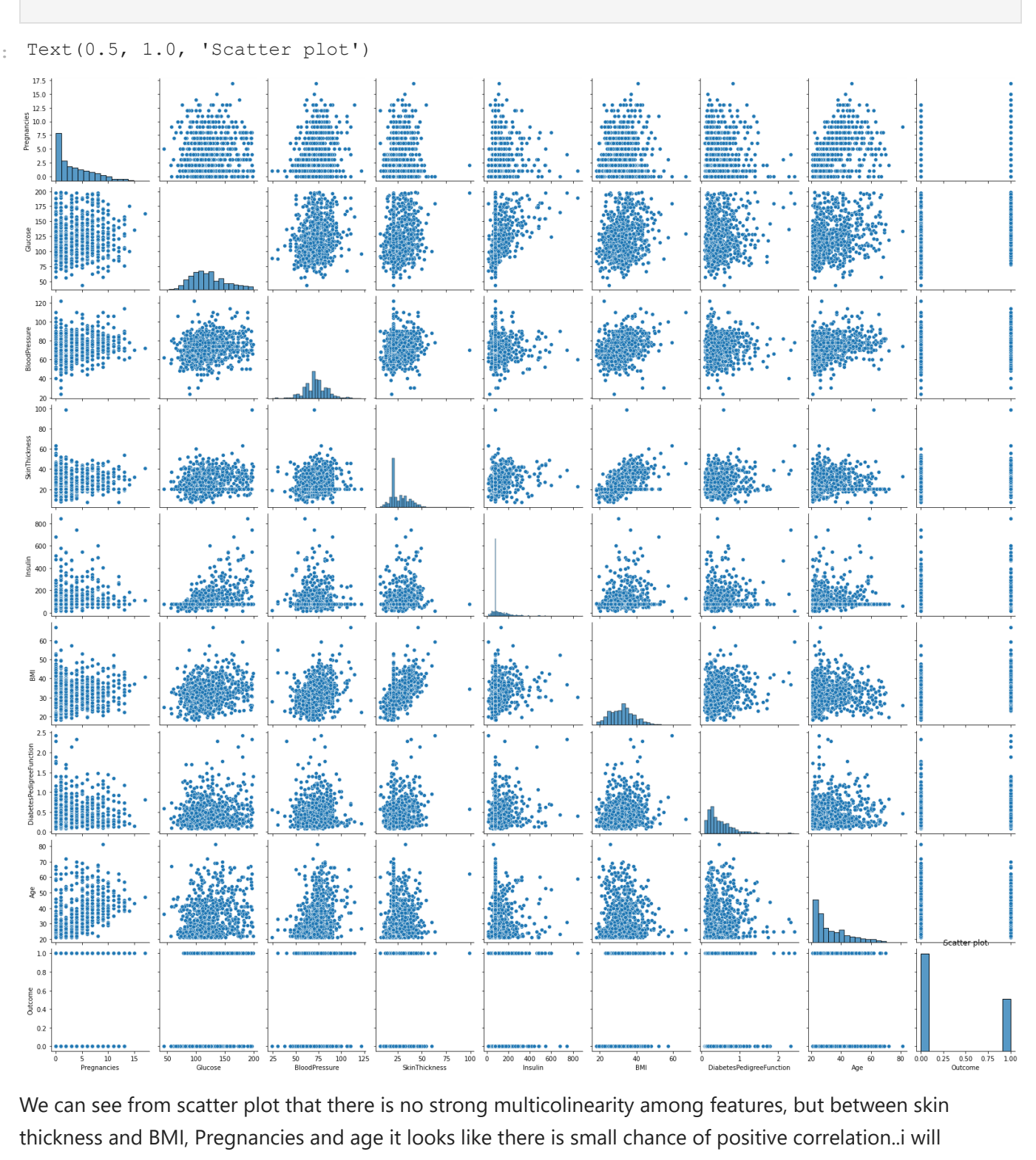
```
Count of class is:
 0    500
1    268
Name: Outcome, dtype: int64
```



We can see that both class is balanced so we need not to perform any sampling method to maintain the balance between both classes. Therefore i will be directly using this data in training and testing purpose without performing any sampling method. Meanwhile during Model Validation , we also need not worry abour ROC Curve because data is not imbalanced, but as this is a medical data so i will be using ROC curve to make sure TYPE 2 ERROR will not be there.

In [28]:
```
sns.pairplot(df)
plt.title('Scatter plot')
```

Out[28]:
```
Text(0.5, 1.0, 'Scatter plot')
```



We can see from scatter plot that there is no strong multicolinearity among features, but between skin thickness and BMI, Pregnancies and age it looks like there is small chance of positive correlation.i will explore more when analyzing correlation

correlation analysis

In [29]:
```
df.corr()
```

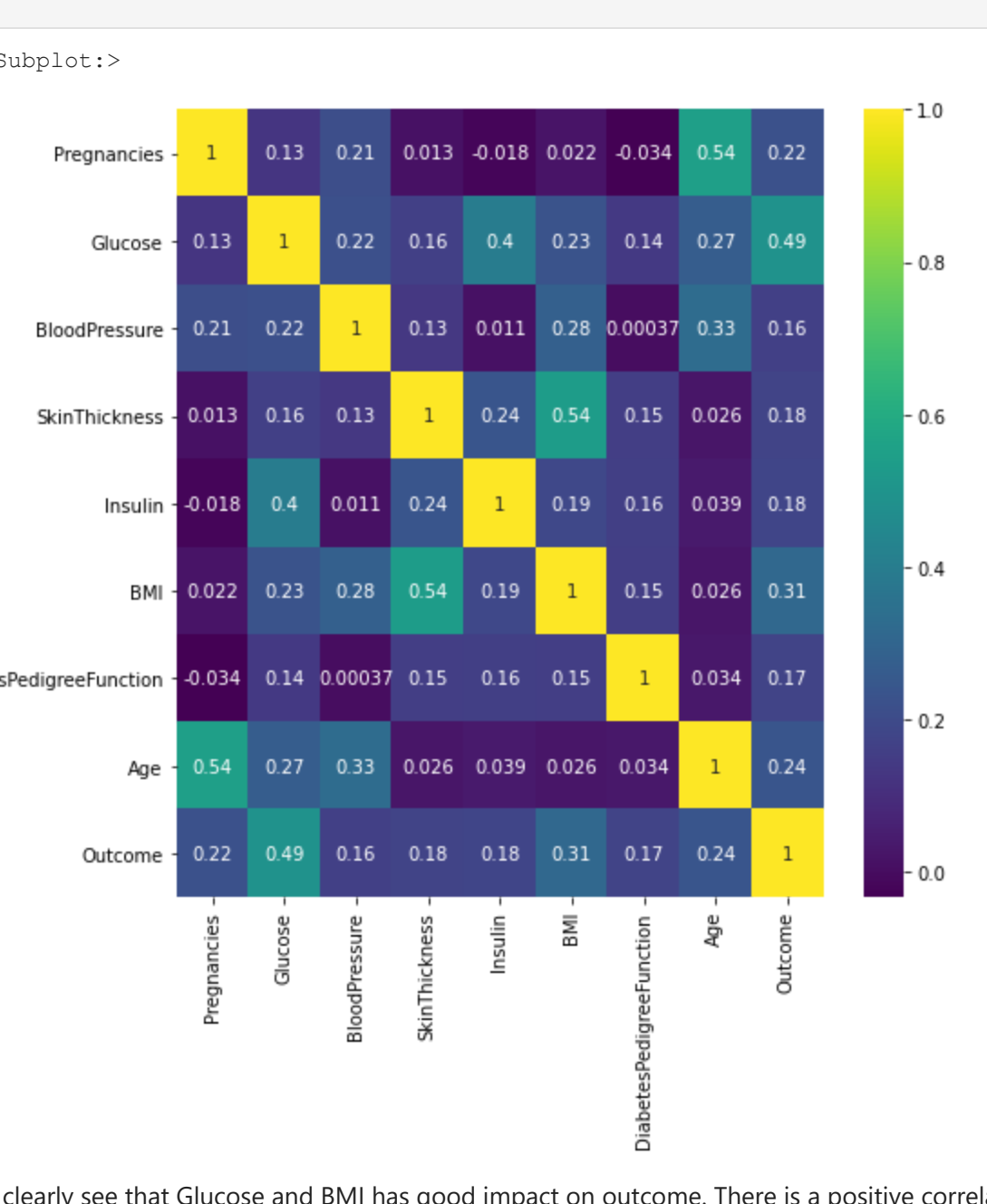
Out[29]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
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	
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	

We can clearly see that Glucose and BMI has good impact on outcome when compared to other factors. There is a positive correlation between BMI and Skinthickness or Pregnancies and age.

In [30]:
```
plt.subplots(figsize=(8,8))
sns.heatmap(df.corr(),annot=True,cmap='viridis')   ### gives correlation value
```

Out[30]:
```
<AxesSubplot:>
```



We can clearly see that Glucose and BMI has good impact on outcome. There is a positive correlation between BMI and Skinthickness or Pregnancies and age

Data Preprocessing

In [31]:
```
x=df.iloc[:,:-1].values
y=df.iloc[:,-1].values
```

In [32]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=0)
```

In [33]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(614, 8)
(154, 8)
(614,)
(154,)
```

data scaling using "StandardScaler" class

In [34]:
```
from sklearn.preprocessing import StandardScaler
Scale=StandardScaler()
x_train_std=Scale.fit_transform(x_train)
x_test_std=Scale.transform(x_test)
```

KNN

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

In [35]:
```
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
classifier_KNN = KNeighborsClassifier(n_neighbors = 25)
#Using 25 Neighbors just as thumb rule sqrt of observation
classifier_KNN.fit(x_train_std,y_train)
knn_pred=classifier_KNN.predict(x_test_std)
```

In [36]:
```
from sklearn.metrics import confusion_matrix
cm_KNN=confusion_matrix(y_test,knn_pred,labels=[1,0])
cm_KNN
```

Out[36]:
```
array([[30, 17],
       [11, 96]], dtype=int64)
```

In [37]:
```
from sklearn.metrics import accuracy_score
KNN_acc = accuracy_score(knn_pred,y_test)
KNN_acc
```

Out[37]:
```
0.8181818181818182
```

In [38]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(knn_pred,y_test)
```

Out[38]:
```
0.7906324195985321
```

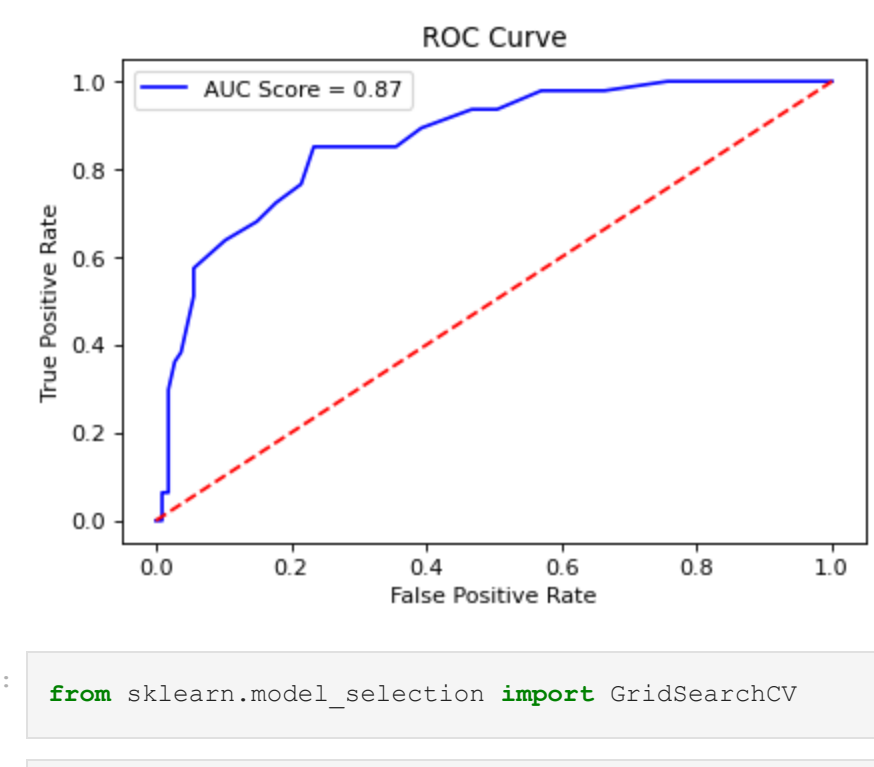
In [39]:
```
import seaborn as sns
sns.heatmap(cm_KNN, annot = True)
```

Out[39]:
```
<AxesSubplot:>
```

In [40]:
```
print("Model Validation ==>\n")
print("\n","Classification Report::")
print(metrics.classification_report(y_test,knn_pred),'\n')
print("\n","ROC Curve")
knn_prob=classifier_KNN.predict_proba(x_test_std)
knn_prob1=knn_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,knn_prob1)
roc_auc_knn=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_knn)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Model Validation ==>


 Classification Report::
              precision    recall  f1-score   support

           0       0.85      0.90      0.87       107
           1       0.73      0.64      0.68        47

    accuracy                           0.82       154
   macro avg       0.79      0.77      0.78       154
weighted avg       0.81      0.82      0.81       154



 ROC Curve
```

Out[40]:
```
<matplotlib.legend.Legend at 0x20fa20aa8b0>
```



In [41]:
```
from sklearn.model_selection import GridSearchCV
```

In [42]:
```
pGrid = {'n_neighbors': range(10,200),
        'leaf_size': range(10, 51, 10), }
gscv = GridSearchCV(estimator = KNeighborsClassifier(), param_grid = pGrid, cv = 5,
                    scoring = 'recall', n_jobs = -1, verbose = True)
```

In [43]:
```
gscv.fit(x,y)
```

```
Fitting 5 folds for each of 950 candidates, totalling 4750 fits
```

Out[43]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'leaf_size': range(10, 51, 10),
                         'n_neighbors': range(10, 200)},
             scoring='recall', verbose=True)
```

In [44]:
```
gscv.best_params_
```

Out[44]:
```
{'leaf_size': 10, 'n_neighbors': 13}
```

In [45]:
```
#Using 13 Neighbors
classifier_KNN1 = KNeighborsClassifier(n_neighbors = 13)
classifier_KNN1.fit(x_train_std,y_train)
knn_pred1=classifier_KNN1.predict(x_test_std)
```

In [46]:
```
cm_KNN1=confusion_matrix(y_test,knn_pred1,labels=[1,0])
cm_KNN1
```

Out[46]:
```
array([[29, 18],
       [10, 97]], dtype=int64)
```

In [47]:
```
KNN_acc1 = accuracy_score(knn_pred1,y_test)
KNN_acc1
```

Out[47]:
```
0.8181818181818182
```

In [48]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(knn_pred1,y_test)
```

Out[48]:
```
0.7935340022296544
```

No difference in KNN model performance with change in n_neighbors parameter

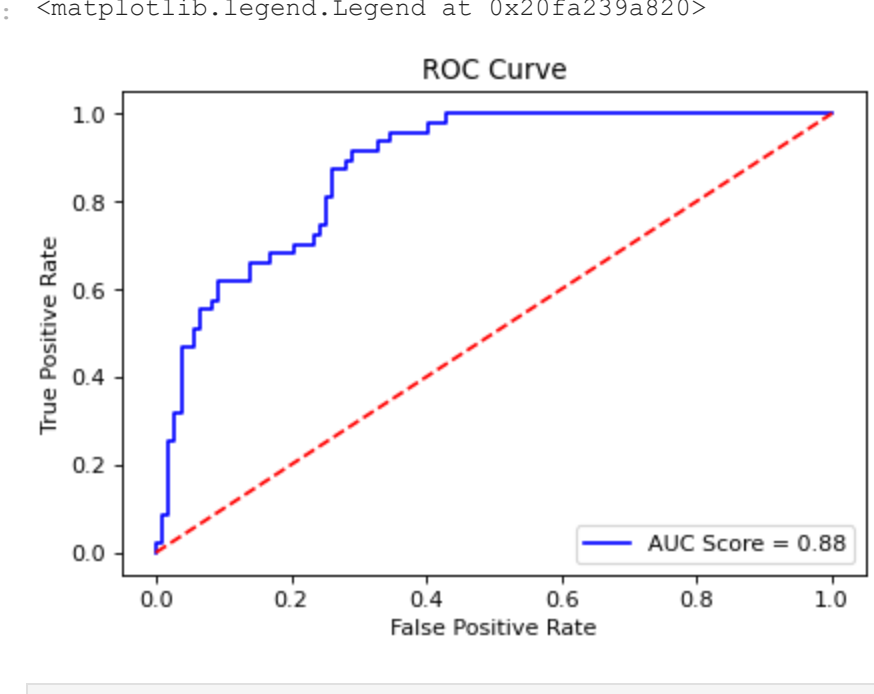
Support Vector Classifier

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

In [49]:
```
from sklearn.svm import SVC
svc_model_linear = SVC(kernel='linear',random_state=0,probability=True,C=0.01)
svc_model_linear.fit(x_train_std,y_train)
svc_pred=svc_model_linear.predict(x_test_std)
svc_acc=accuracy_score(svc_pred,y_test)
matrix_SVC=confusion_matrix(y_test,svc_pred,labels=[1,0])
print('Confusion matrix : \n',matrix_SVC)
print("\n" "Model Validation ==>\n")
print("Accuracy Score of SVC Model with Linear Kernel::")
print(metrics.accuracy_score(y_test,svc_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,svc_pred),'\n')
print("\n","ROC Curve")
svc_prob_linear=svc_model_linear.predict_proba(x_test_std)
svc_prob_linear1=svc_prob_linear[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,svc_prob_linear1)
roc_auc_svc=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_svc)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Confusion matrix :
 [[27 20]
 [ 9 98]]

Model Validation ==>

Accuracy Score of SVC Model with Linear Kernel::
0.8116883116883117

 Classification Report::
              precision    recall  f1-score   support

           0       0.83      0.92      0.87       107
           1       0.75      0.57      0.65        47

    accuracy                           0.81       154
   macro avg       0.79      0.75      0.76       154
weighted avg       0.81      0.81      0.80       154



 ROC Curve
```

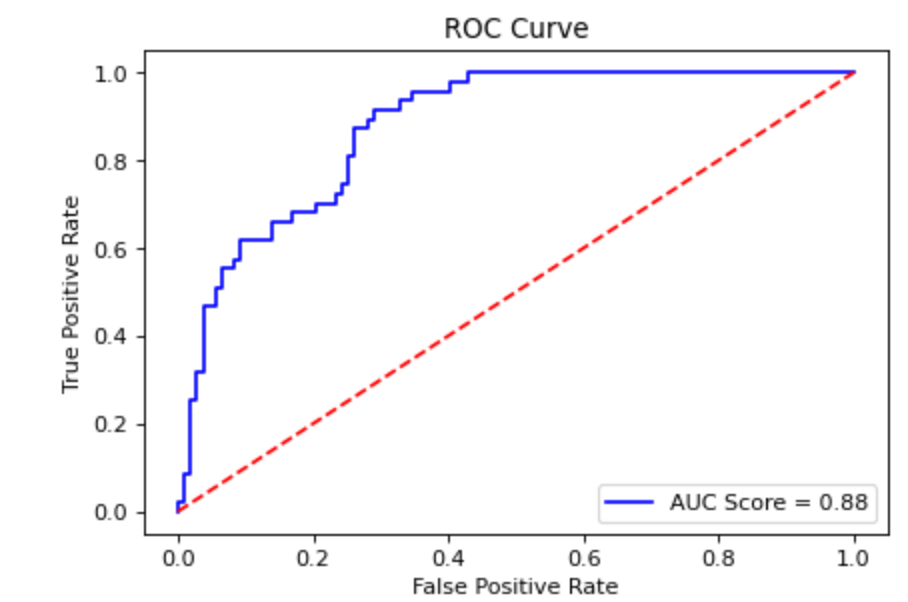
Out[49]:
```
<matplotlib.legend.Legend at 0x20fa239a820>
```



In [50]:
```
from sklearn.svm import SVC
svc_model_rbf = SVC(kernel='rbf',random_state=0,probability=True,C=1)
svc_model_rbf.fit(x_train_std,y_train)
svc_pred_rbf=svc_model_rbf.predict(x_test_std)
matrix_SVM_rbf=confusion_matrix(y_test,svc_pred_rbf,labels=[1,0])
print('Confusion matrix : \n',matrix_SVM_rbf)
print("\n" "Model Validation ==>\n")
print("Accuracy Score of SVC Model with RBF Kernel::")
print(metrics.accuracy_score(y_test,svc_pred_rbf))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,svc_pred_rbf),'\n')
print("\n","ROC Curve")
svc_prob_rbf=svc_model_linear.predict_proba(x_test_std)
svc_prob_rbf1=svc_prob_rbf[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,svc_prob_rbf1)
roc_auc_svc=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_svc)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Confusion matrix :
 [[25 22]
 [13 94]]

 Model Validation ==>

Accuracy Score of SVC Model with RBF Kernel::
0.7727272727272727

 Classification Report::
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       107
           1       0.66      0.53      0.59        47

    accuracy                           0.77       154
   macro avg       0.73      0.71      0.72       154
weighted avg       0.76      0.77      0.77       154



 ROC Curve
```

Out[50]:
```
<matplotlib.legend.Legend at 0x20fa2181280>
```



SVC with Linear Kernel is better than RBF Kernel, This was actually expected beause variables are somewhat depending linearly with outcome

Comparing with KNN

Both Models are working fine , but SVC Linear with C=0.01 is better in terms of AUC Score

Logistic Regression

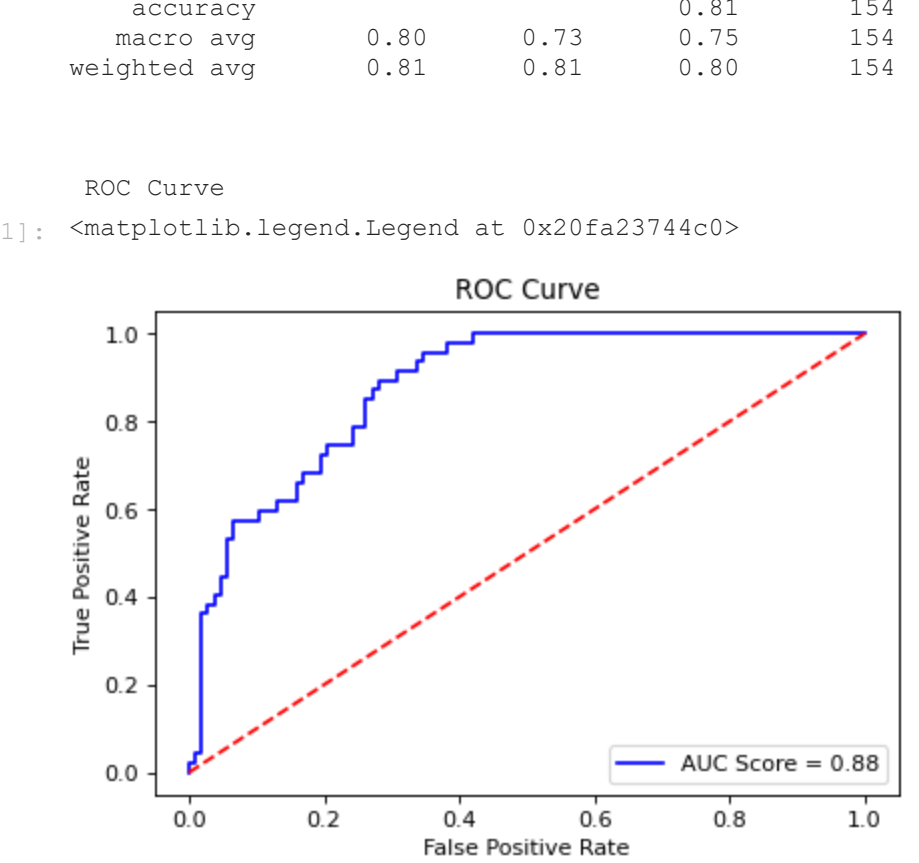
Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression)

In [51]:

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(C=0.01)
lr_model.fit(x_train_std,y_train)
lr_pred=lr_model.predict(x_test_std)
lr_acc=accuracy_score(lr_pred,y_test)
matrix_lr=confusion_matrix(y_test,lr_pred,labels=[1,0])
print("Confusion matrix : \n",matrix_lr)
print("\n","Model Validation ==>\n")
print("Accuracy Score of Logistic Regression Model::")
print(metrics.accuracy_score(y_test,lr_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,lr_pred),'\n')
print("\n","ROC Curve")
lr_prob=lr_model.predict_proba(x_test_std)
lr_prob1=lr_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,lr_prob1)
roc_auc_lr=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label="AUC Score = %0.2f"%roc_auc_lr)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Confusion matrix :
 [[ 25  22]
 [  7 100]]

 Model Validation ==>

Accuracy Score of Logistic Regression Model::
0.8116883116883117

 Classification Report::
              precision    recall  f1-score   support

           0       0.82      0.93      0.87       107
           1       0.78      0.53      0.63        47

    accuracy                           0.81       154
   macro avg       0.80      0.73      0.75       154
weighted avg       0.81      0.81      0.80       154


 ROC Curve
```

Out[51]: <matplotlib.legend.Legend at 0x20fa23744c0>



Accuracy of KNN is better than Logistic Regression,but auc score of Logistic regression is better

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

Ensemble Learning(RF)

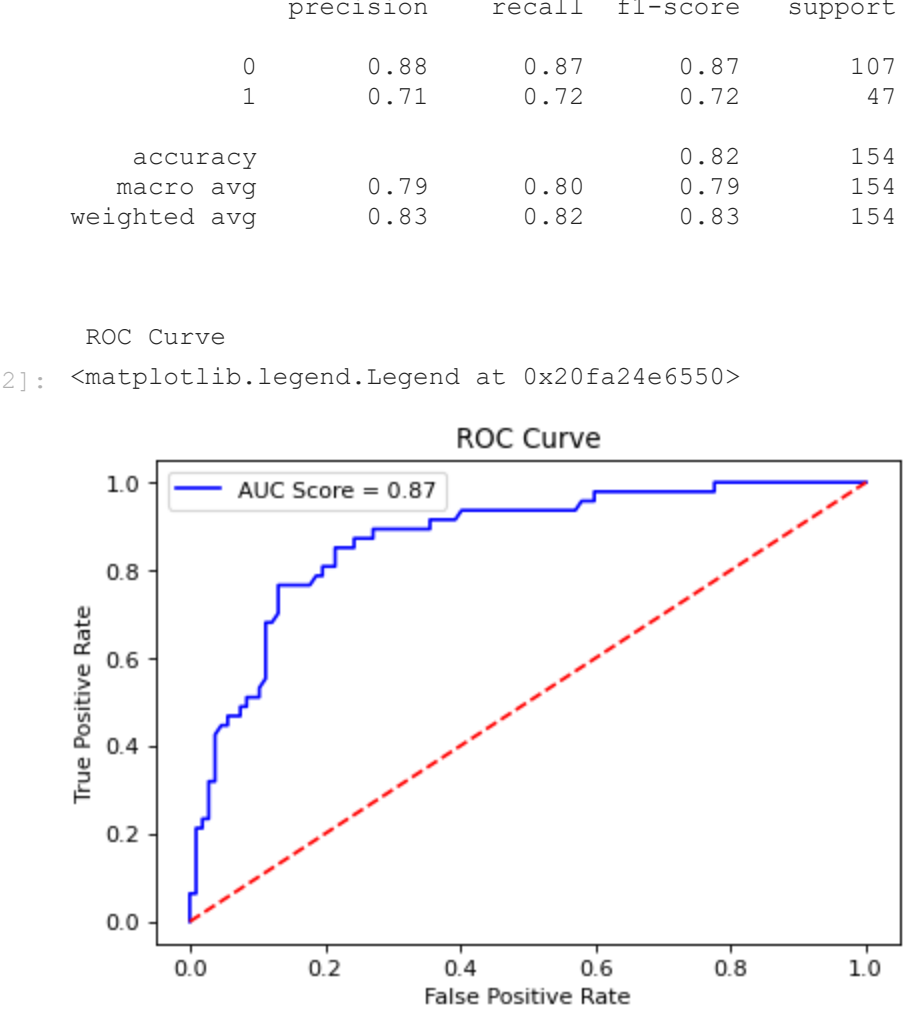
Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean/average prediction of the individual trees.

In [52]:

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=1000,random_state=0)
rf_model.fit(x_train_std,y_train)
rf_pred=rf_model.predict(x_test_std)
rf_acc=accuracy_score(y_test,rf_pred)
matrix_rf=confusion_matrix(y_test,rf_pred,labels=[1,0])
print("Confusion matrix : \n",matrix_rf)
print("\n","Model Validation ==>\n")
print("Accuracy Score of Logistic Regression Model::")
print(metrics.accuracy_score(y_test,rf_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,rf_pred),'\n')
print("\n","ROC Curve")
rf_prob=rf_model.predict_proba(x_test_std)
rf_prob1=rf_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,rf_prob1)
roc_auc_rf=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label="AUC Score = %0.2f"%roc_auc_rf)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Confusion matrix :
 [[34 13]
 [14 93]]

 Model Validation ==>

Accuracy Score of Logistic Regression Model::
0.8246753246753247

 Classification Report::
              precision    recall  f1-score   support

           0       0.88      0.87      0.87       107
           1       0.71      0.72      0.72        47

    accuracy                           0.82       154
   macro avg       0.79      0.80      0.79       154
weighted avg       0.83      0.82      0.83       154


 ROC Curve
```

Out[52]: <matplotlib.legend.Legend at 0x20fa24e6550>



Decision tree

A decision tree 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 [53]:

```
#importing library for Descision Tree and we have two criterion for DT 'gini' and 'En
# you can use gini when there is binary classification otherwise use entropy
from sklearn.tree import DecisionTreeClassifier
#making instance of DT
classifier = DecisionTreeClassifier(criterion = 'gini', random_state = 0,
                                    max_depth = 2, min_samples_leaf = 10, min_samples_
                                    )
classifier.fit(x_train, y_train)
```

Out[53]: DecisionTreeClassifier(max_depth=2, min_samples_leaf=10, min_samples_split=20, random_state=0)

In [54]:

```
#prediction using 'testing' data
y_pred = classifier.predict(x_test)
```

In [55]:

```
#checking score via importing library
from sklearn.metrics import accuracy_score
DT_acc = accuracy_score(y_pred,y_test)
DT_acc
```

Out[55]: 0.7597402597402597

In [56]:

```
cm_DT = confusion_matrix(y_test, y_pred)
cm_DT
```

Out[56]: array([[93, 14],
       [23, 24]], dtype=int64)

In [57]:

```
#checking Auc_Roc_Score via importing library
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred)
```

Out[57]: 0.6898985881885066

NAIVEBYES

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

In [58]:

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
y_pred_nb = classifier.predict(x_test)
NB_acc = accuracy_score(y_test,y_pred_nb)
NB_acc
```

Out[58]: 0.7792207792207793

In [59]:

```
roc_auc_score(y_test,y_pred_nb)
```

Out[59]: 0.7277788824816066

In [60]:

```
cm_NB= confusion_matrix(y_test, y_pred_nb)
cm_NB
```

Out[60]: array([[92, 15],
       [19, 28]], dtype=int64)

In [61]:

```
lis=['KNN','SVM','Logistic Regression','Decision Tree','Random Forest','Naive bayes']
accuracy = KNN_acc,svc_acc,lr_acc,DT_acc,rf_acc,NB_acc
Scores=pd.DataFrame({'MODELS':lis,'ACCURACY':accuracy})
Scores
```

Out[61]:

	MODELS	ACCURACY
0	KNN	0.818182
1	SVM	0.811688
2	Logistic Regression	0.811688
3	Decision Tree	0.759740
4	Random Forest	0.824675
5	Naive bayes	0.779221

In [62]:

```
Scores[(Scores['ACCURACY']) == max(Scores['ACCURACY'])]
```

Out[62]:

	MODELS	ACCURACY
4	Random Forest	0.824675

SO RANDOM FOREST is best model for classification of given dataset