In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
from sklearn import metrics
```

In [2]:
```
df = pd.read_csv('C:/Users/ramas/Desktop/Data Science Simplilearn/Data Science Capst
df.head()
```

Out[2]:

	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

Descriptive Analysis

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
```

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

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
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	

Insights from Descriptive Analysis

There is 768 observations of 9 variable. Independent variables are Pregnencies , Glucose, BloodPressure, Insulin, BMI and DiabetesPedigree Function. Age is Outcome Variable. Average Age of Patients are 33.24 with minimum being 21 and maximum 81. 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 . Variation in variables can be easily observed from table below :->

In [5]:
```
print("Standard Deviation of each variables are ==> ")
df.apply(np.std)
```

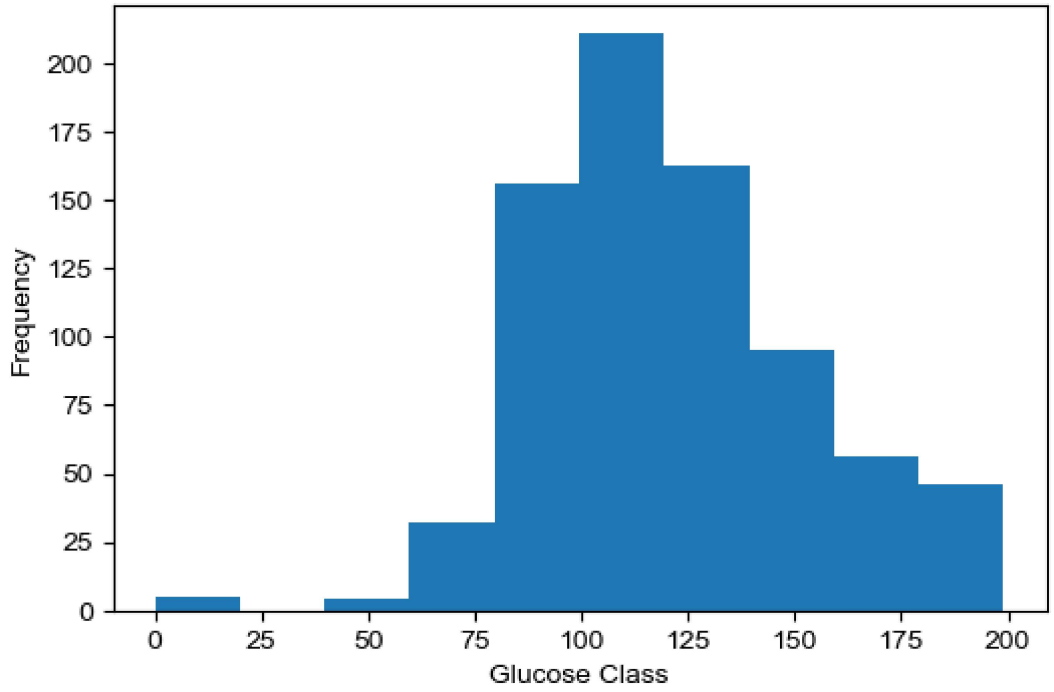
```
Standard Deviation of each variables are ==>
```

Out[5]:
```
Pregnancies                 3.367384
Glucose                    31.951796
BloodPressure              19.343202
SkinThickness              15.941829
Insulin                   115.168949
BMI                         7.879026
DiabetesPedigreeFunction    0.331113
Age                        11.752573
Outcome                     0.476641
dtype: float64
```

Treating Missing Values and Analysing Distribution of Data

In [6]:
```
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel('Glucose Class')
df['Glucose'].plot.hist()
sns.set_style(style='darkgrid')
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
```

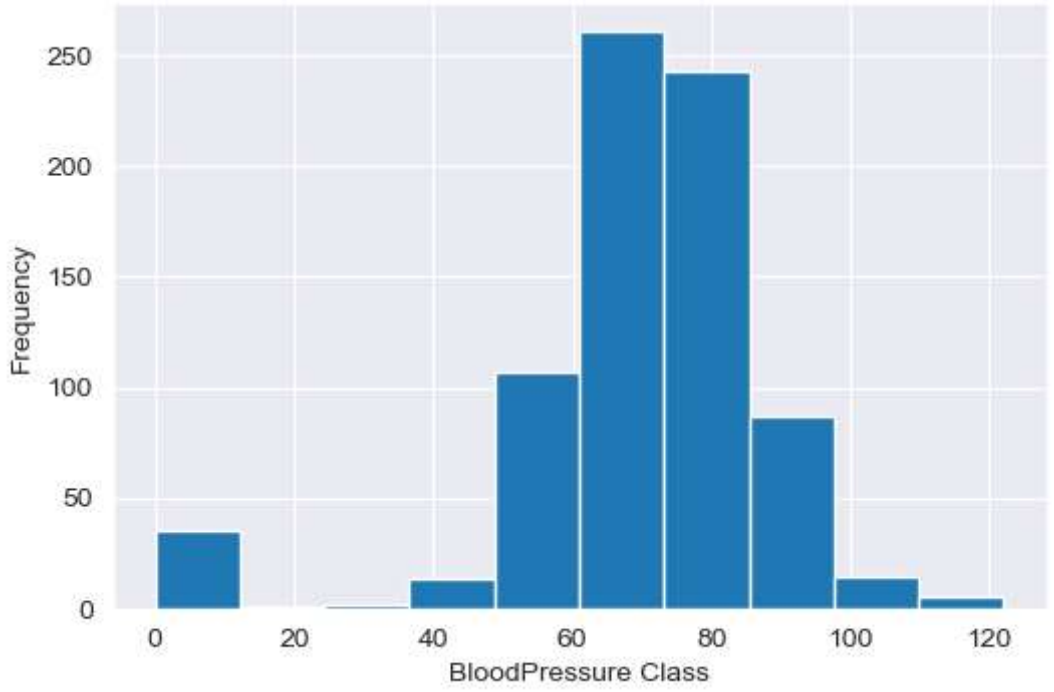


I am treating missing values which is basically 0 by mean of Glucose level. This is because we can see from histogram most of observation have Glucose level between 100 and 120.

In [7]:
```
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
```

In [8]:
```
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel('BloodPressure Class')
df['BloodPressure'].plot.hist()
sns.set_style(style='darkgrid')
print("Mean of BloodPressure level is :-", df['BloodPressure'].mean())
print("Datatype of BloodPressure Variable is:",df['BloodPressure'].dtypes)
```

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



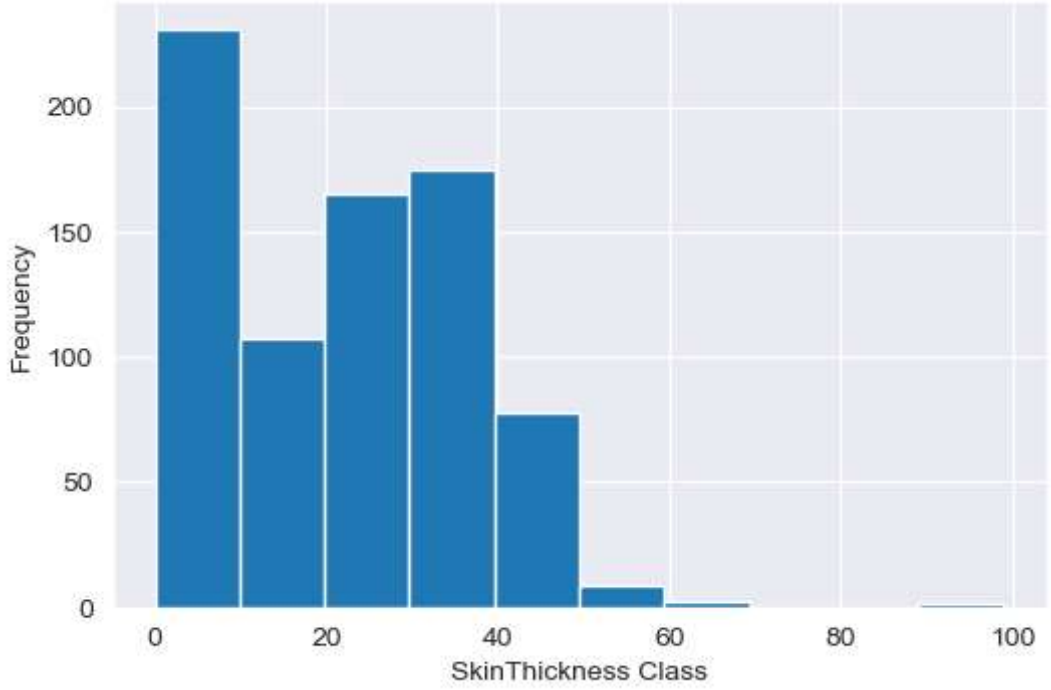
I am treating missing values which is basically 0 by mean of BloodPressure level. This is

because we can see from histogram most of observation have BP level between 70 and 80.

In [9]:
```
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())
```

In [10]:
```
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel('SkinThickness Class')
df['SkinThickness'].plot.hist()
sns.set_style(style='darkgrid')
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
```

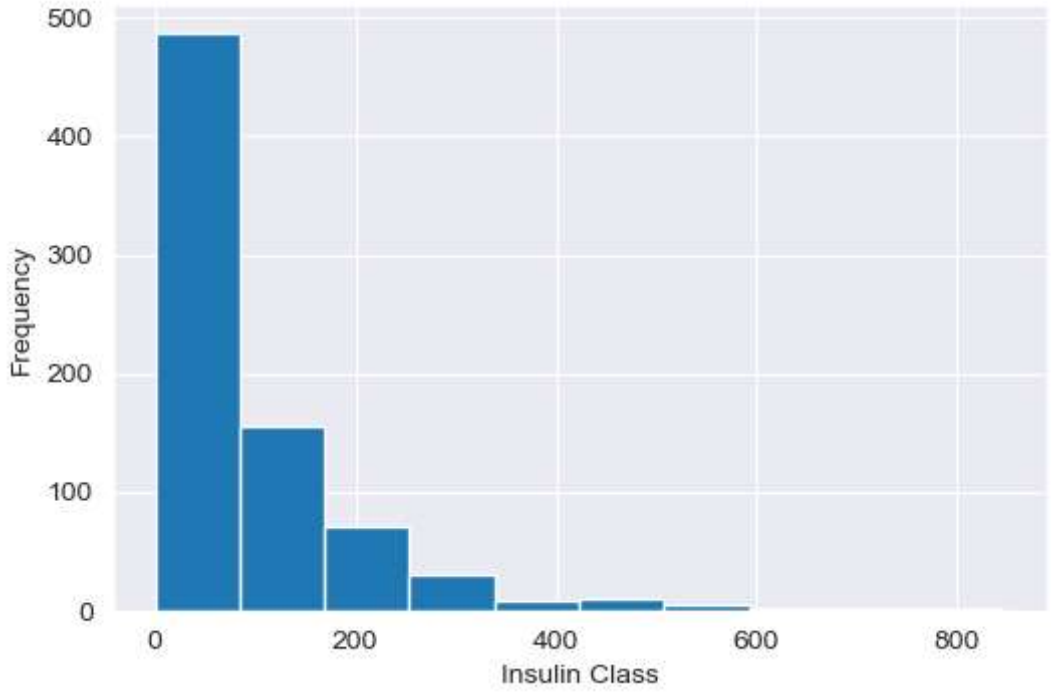


I am treating missing values which is basically 0 by mean of SkinThickness. This is because we can see from histogram most of observation have SkinThickness between 20 and 30.

In [11]:
```
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].mean())
```

In [12]:
```
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel('Insulin Class')
df['Insulin'].plot.hist()
sns.set_style(style='darkgrid')
print("Mean of Insulin is :-", df['Insulin'].mean())
print("Datatype of Insulin Variable is:",df['Insulin'].dtypes)
```

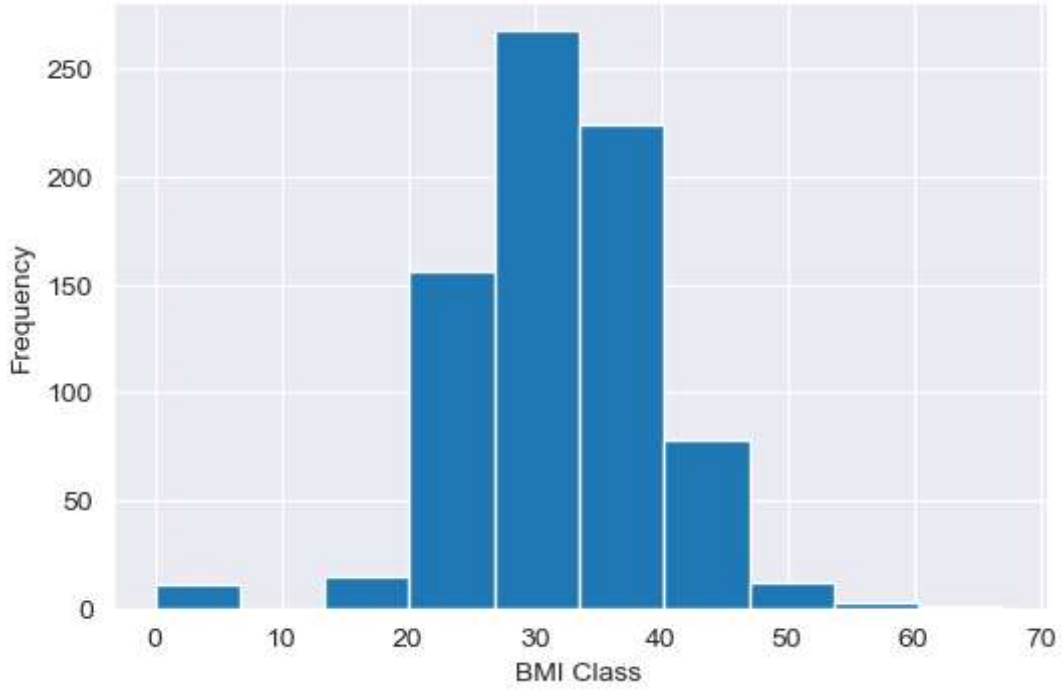
```
Mean of Insulin is :- 79.79947916666667
Datatype of Insulin Variable is: int64
```



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

In [14]:
```
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel('BMI Class')
df['BMI'].plot.hist()
sns.set_style(style='darkgrid')
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
```

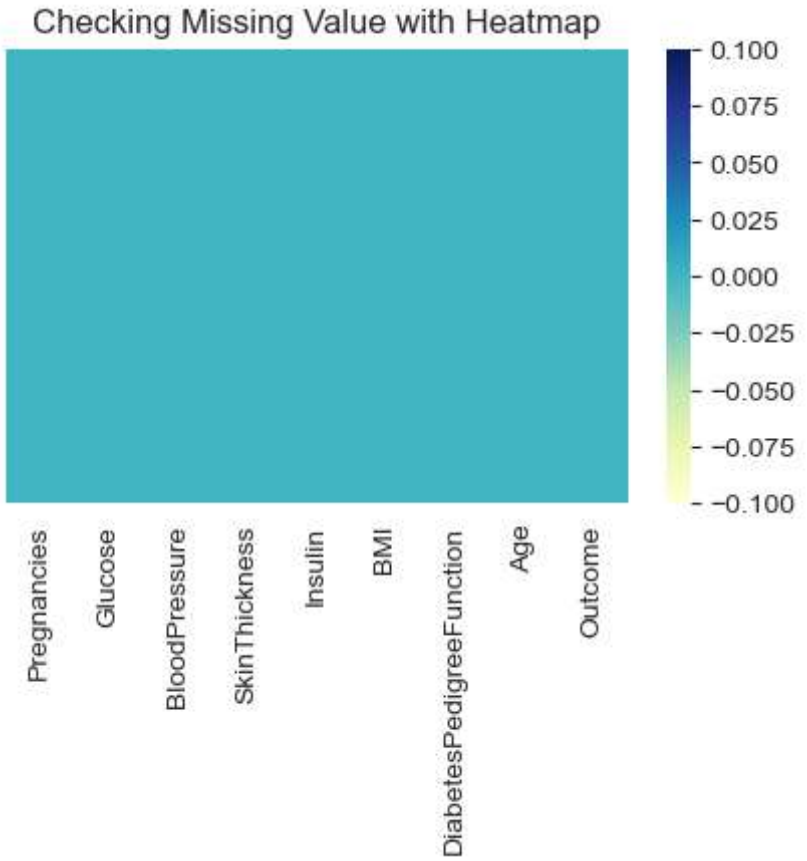


In [15]:
```
df['BMI']=df['BMI'].replace(0,df['BMI'].mean())
```

In [16]:
```

```

```
plt.figure(figsize=(5,3),dpi=100)
plt.title('Checking Missing Value with Heatmap')
sns.heatmap(df.isnull(),cmap='YlGnBu',yticklabels=False)
```

Out[16]: <AxesSubplot:title={'center':'Checking Missing Value with Heatmap'}>



In [17]:
```
df.head()
```

Out[17]:

	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

In [18]:
```
df.tail()
```

Out[18]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
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

In [19]:
```
df.to_csv('after_week1.csv',index=False)
```

Exploratory Data Analysis

In [20]:
```
df2=pd.read_csv('after_week1.csv')
df2.head()
```

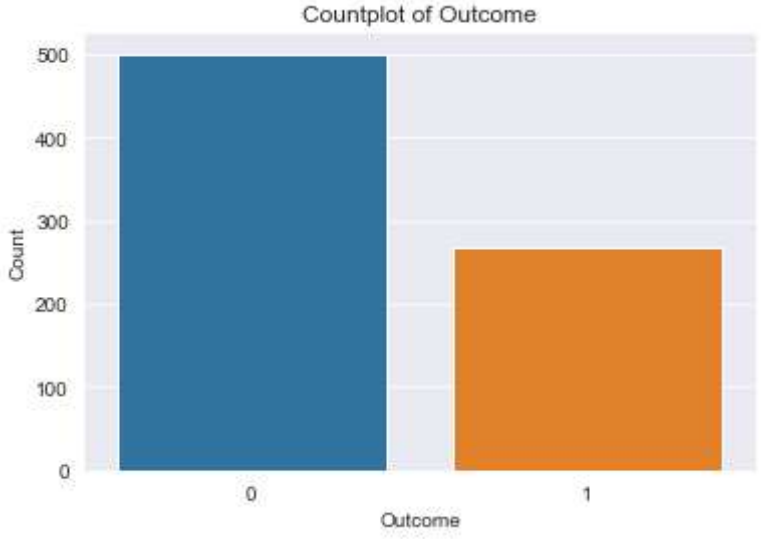
Out[20]:

	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

Countplot

In [21]:
```
sns.set_style('darkgrid')
sns.countplot(df['Outcome'])
plt.title("Countplot of Outcome")
plt.xlabel('Outcome')
plt.ylabel("Count")
print("Count of class is:\n",df['Outcome'].value_counts())
```

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

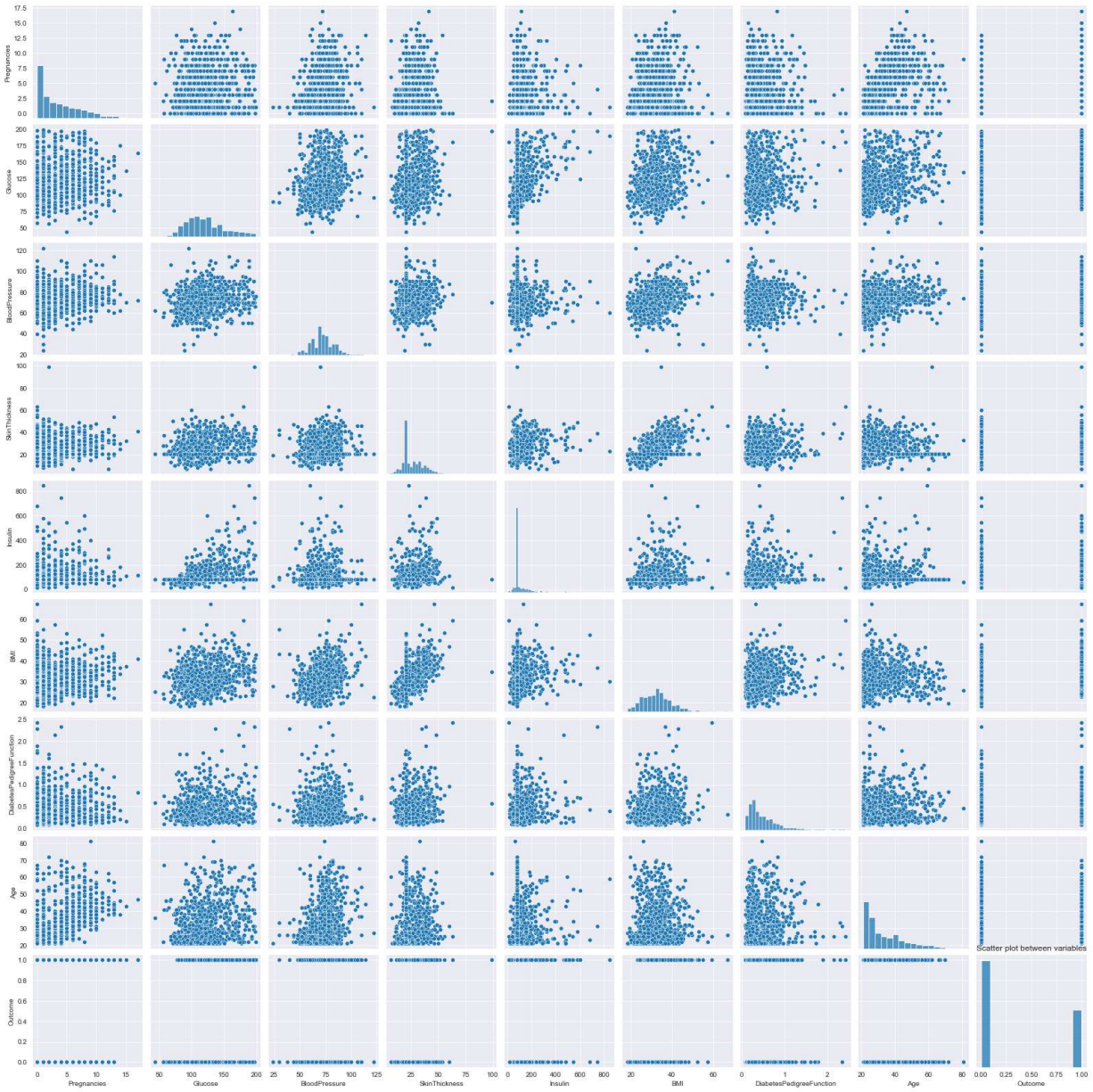


We can see that both class is balanced so we need not to perform any sampling method to maintain the balance between both classes. Therefor i will be directly using this data in training and testing purpose without performing any sampling method. Meanwhile during Model Validation , we also need not worry 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.

Scatter Plot

In [22]:
```
sns.pairplot(df)
plt.title('Scatter plot between variables')
```

Out[22]:
```
Text(0.5, 1.0, 'Scatter plot between variables')
```



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

Corelation Analysis

In [23]:
```
df2.corr()
```

Out[23]:

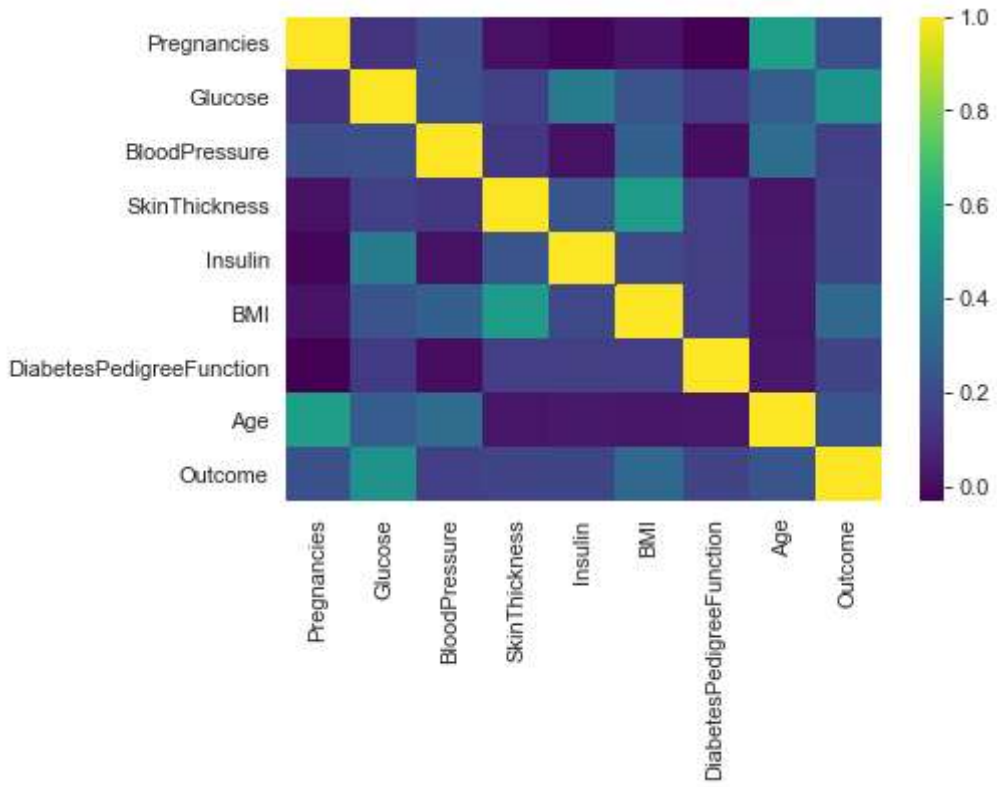
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
Pregnancies	1.000000	0.127964	0.208984	0.013376	-0.018082	0.021546
Glucose	0.127964	1.000000	0.219666	0.160766	0.396597	0.231478
BloodPressure	0.208984	0.219666	1.000000	0.134155	0.010926	0.281231

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
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. There is a strong positive correlation between BMI and Skinthickness or Pregnancies and age

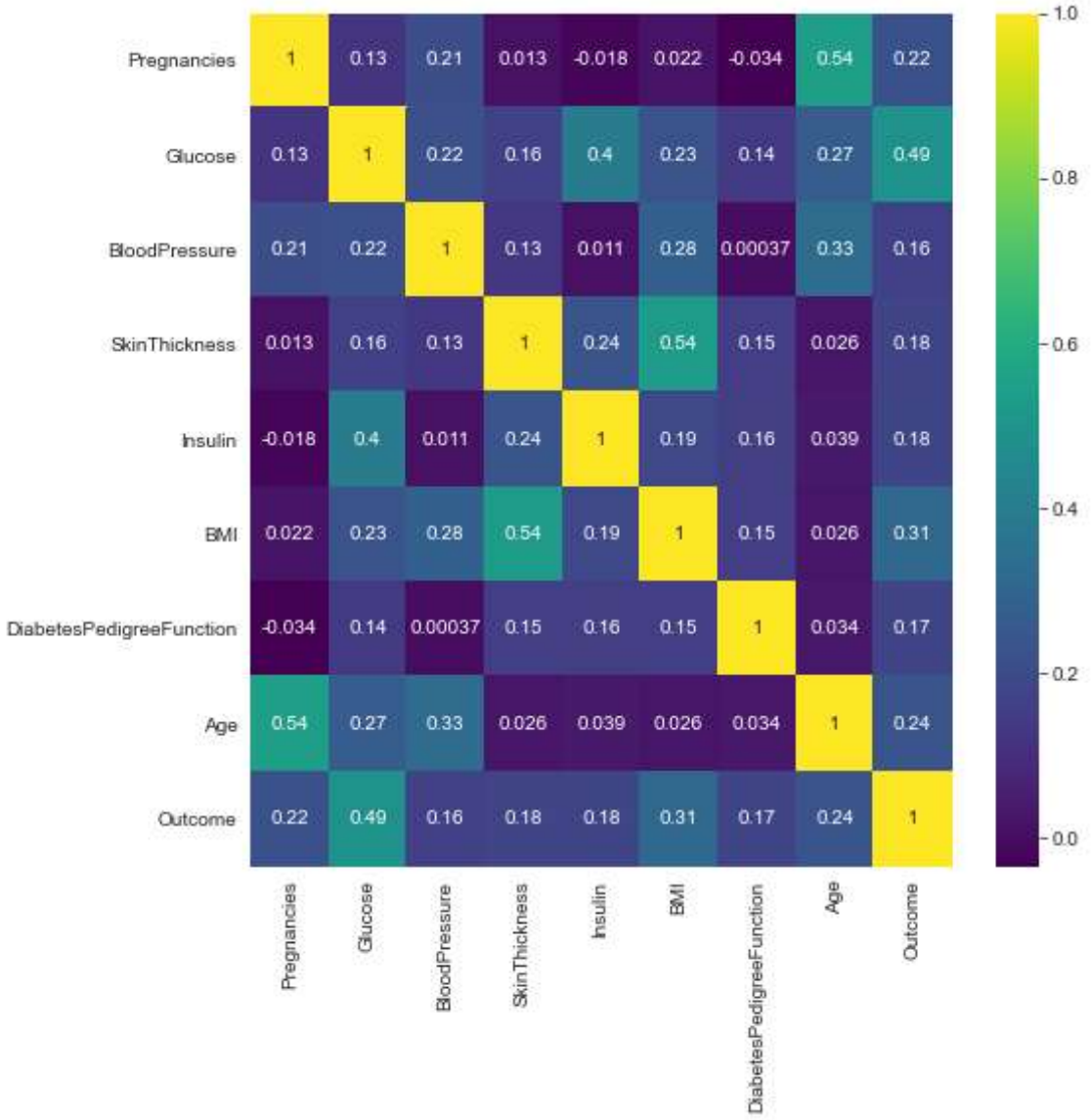
In [24]:
```
plt.figure(dpi=80)
sns.heatmap(df.corr(),cmap='viridis')
```

Out[24]: <AxesSubplot:>



In [25]:
```
plt.subplots(figsize=(8,8))
sns.heatmap(df2.corr(),annot=True,cmap='viridis')
```

Out[25]: <AxesSubplot:>



Data Preprocessing

In [26]:
```
x=df2.iloc[:,:-1].values
y=df2.iloc[:,-1].values
```

In [27]:
```
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 [28]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

In [29]:
```
from sklearn.preprocessing import StandardScaler
```

In [30]:
```
Scale=StandardScaler()
```

```
x_train_std=Scale.fit_transform(x_train)
x_test_std=Scale.transform(x_test)
```

In [31]:
```
norm=lambda a:(a-min(a))/(max(a)-min(a))
```

In [32]:
```
df2_norm=df2.iloc[:,:-1]
```

In [33]:
```
df2_normalized=df2_norm.apply(norm)
```

In [34]:
```
x_train_norm,x_test_norm,y_train_norm,y_test_norm=train_test_split(df2_normalized.va
```

In [35]:
```
print(x_train_norm.shape)
print(x_test_norm.shape)
print(y_train_norm.shape)
print(y_test_norm.shape)
```

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

Data is mostly numerical and in such scenario , Logistic Regression works fine. We have also seen in week 2 that variables are depending on target somewhat linearly, So this is also good for Logistic Regression. I will be also using Support Vector Classifier, Perceptron Learning, Random Forest (Ensemble Learning) to see if i can improve accuracy. Note these learning algorithm also works on linear data very well. To validate model i will be using train test split, for accuracy i will be using accuracy using confusion matrix because classes are balanced and i will be also considering ROC Curve and ROC AUC Score to make sure Type 2 Error will not occur for Positive class, that is 1

KNN

KNN with Standard Scaling

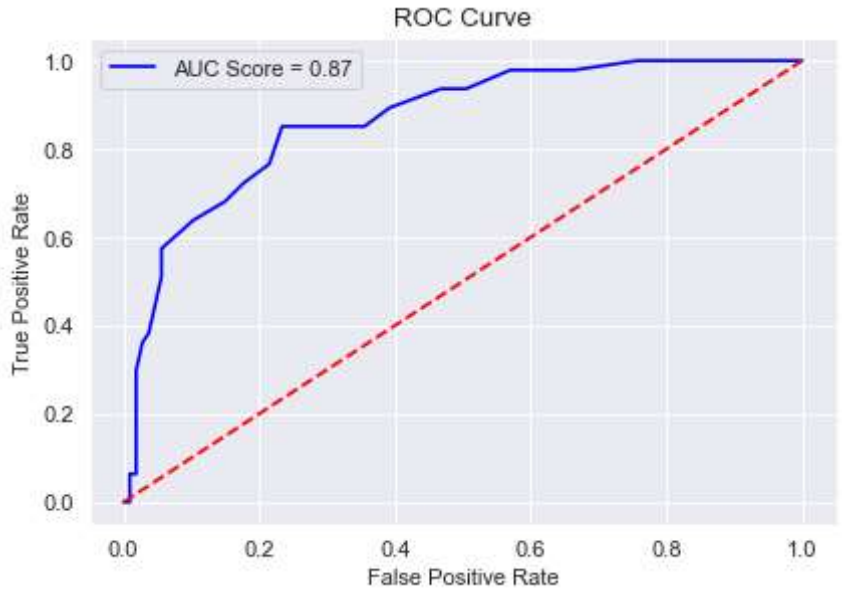
In [36]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=25)
#Using 25 Neighbors just as thumb rule sqrt of observation
knn_model.fit(x_train_std,y_train)
knn_pred=knn_model.predict(x_test_std)
```

In [37]:
```
print("Model Validation ==>\n")
print("Accuracy Score of KNN Model::")
print(metrics.accuracy_score(y_test,knn_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,knn_pred),'\n')
print("\n","ROC Curve")
knn_prob=knn_model.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 ==>

Accuracy Score of KNN Model::
0.8181818181818182

 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[37]: <matplotlib.legend.Legend at 0x1e59b761e50>



KNN with normalization

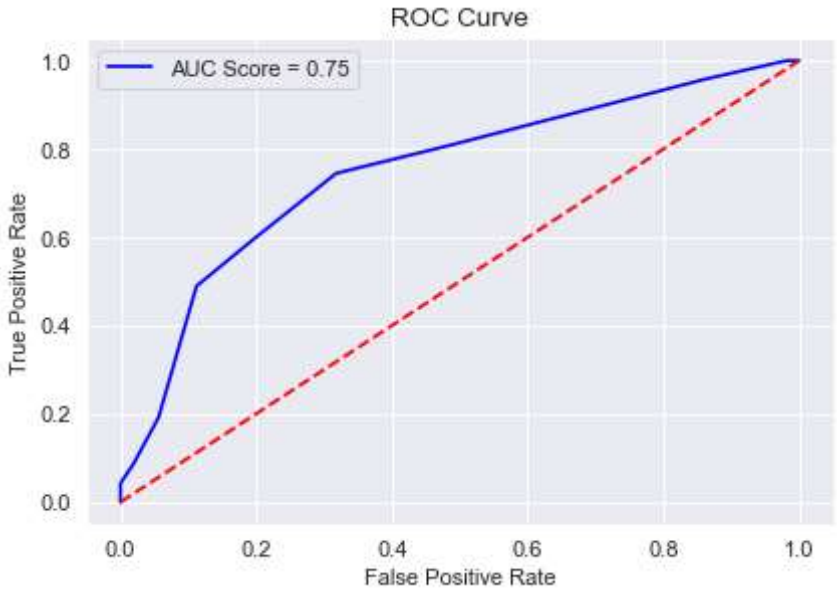
In [38]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model_norm = KNeighborsClassifier(n_neighbors=25)
#Using 25 Neighbors just as thumb rule sqrt of observation
knn_model_norm.fit(x_train_norm,y_train_norm)
knn_pred_norm=knn_model_norm.predict(x_test_norm)
```

In [39]:
```
print("Model Validation ==>\n")
print("Accuracy Score of KNN Model with Normalization::")
print(metrics.accuracy_score(y_test_norm,knn_pred_norm))
print("\n","Classification Report::")
print(metrics.classification_report(y_test_norm,knn_pred_norm),'\n')
print("\n","ROC Curve")
knn_prob_norm=knn_model.predict_proba(x_test_norm)
knn_prob_norm1=knn_prob_norm[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test_norm,knn_prob_norm1)
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 ==>

Accuracy Score of KNN Model with Normalization::
0.8311688311688312

 Classification Report::
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       107
           1       0.74      0.68      0.71        47

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


 ROC Curve
```

Out[39]: <matplotlib.legend.Legend at 0x1e59b7c7be0>



We can clearly see that KNN with Standardization is better than Normalization, So later i will build models using Z Score Standardization and will compare with KNN

Support Vectore Classifier

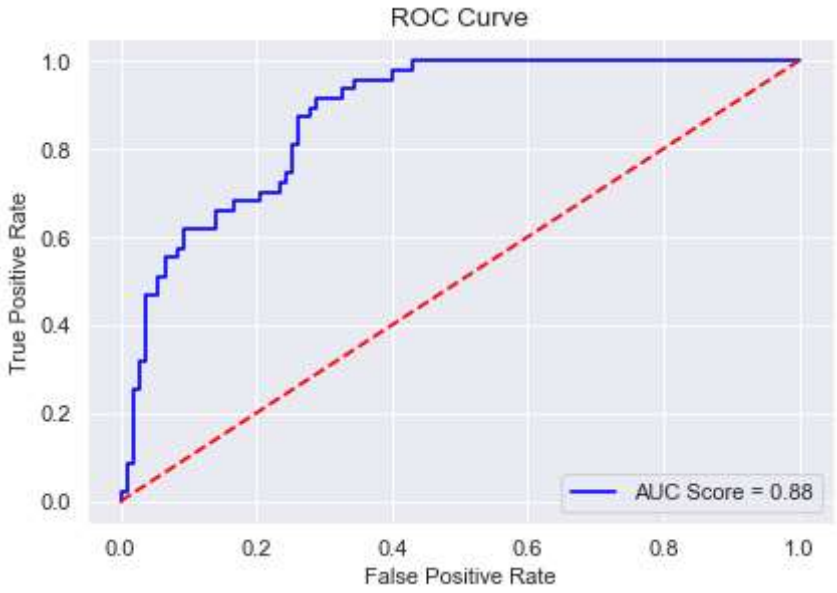
In [40]:
```
from sklearn.svm import SVC
svc_model_linear = SVC(kernel='linear',random_state=0,probability=True,C=0.01)
svc_model_linear.fit(x_train_std,y_train)
svc_pred=svc_model_linear.predict(x_test_std)
```

In [41]:
```
print("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()
```

```
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[41]:
```
<matplotlib.legend.Legend at 0x1e59b831fa0>
```



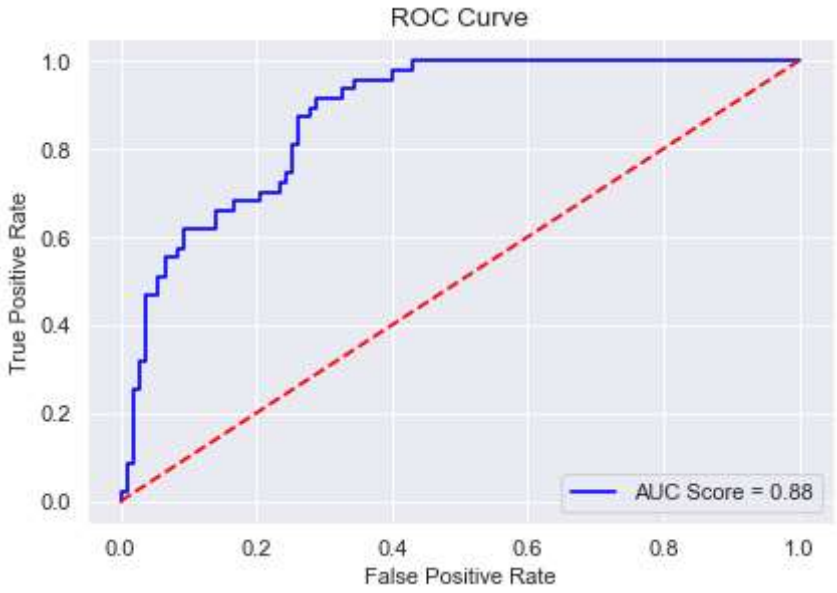
In [42]:
```
from sklearn.svm import SVC
svc_model_rbf = SVC(kernel='rbf',random_state=0,probability=True,C=1)
svc_model_rbf.fit(x_train_std,y_train)
svc_pred_rbf=svc_model_rbf.predict(x_test_std)
```

In [43]:
```
print("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()
```

```
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[43]: <matplotlib.legend.Legend at 0x1e59b8919a0>



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.

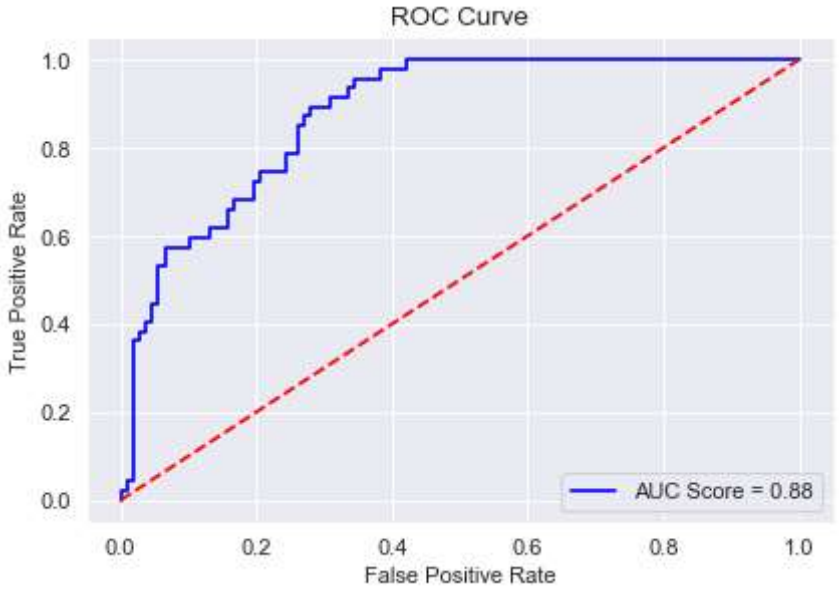
Logistic Regression

In [44]:
```
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)
```

In [45]:
```
print("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()
```

```
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[45]:
```
<matplotlib.legend.Legend at 0x1e59b8ee700>
```



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

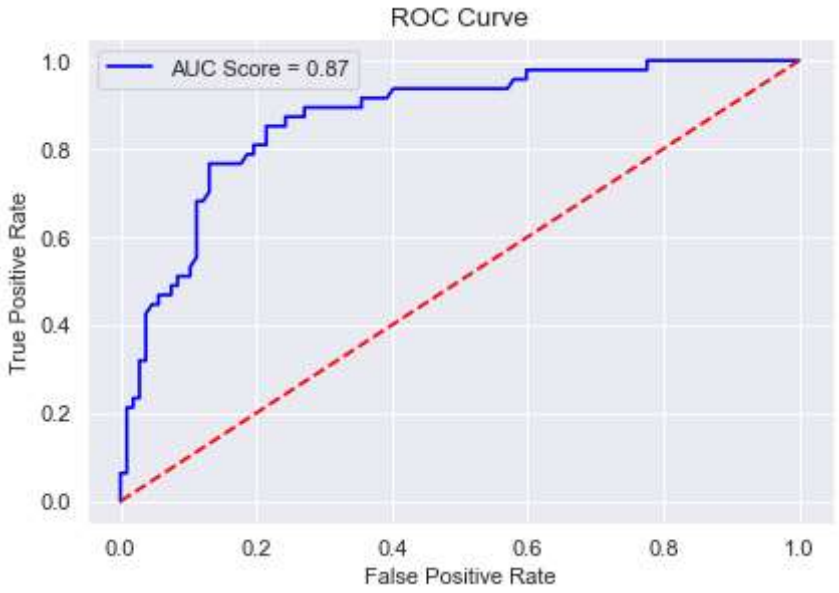
Ensemble Learning(RF)

In [46]:
```
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)
```

In [47]:
```
print("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.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_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()
```

```
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[47]:
```
<matplotlib.legend.Legend at 0x1e59caa1610>
```



So we can see Random Forest Classifier is best among all, you might be wondering auc score is lesser by 1 than others also i am considering it to be best because balance of classes between Precision and Recall is far better than other Models. So we can consider a loss in AUC by 1

In [ ]: