HealthCare_Capstone

August 30, 2021

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

[2]:
```
# Importing the required dataset
data = pd.read_csv("health care diabetes.csv")
```

[3]:
```
data.head()
```

[3]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[4]:
```
# Checking for null values
data.isnull().any()
```

[4]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
```

```
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

1 *Descriptive Analysis*

[5]:
```
data.info()
```

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

[6]:
```
data.describe()
```

[6]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
```

```
50%      32.000000                    0.372500   29.000000    0.000000
75%      36.600000                    0.626250   41.000000    1.000000
max      67.100000                    2.420000   81.000000    1.000000
```

[7]:
```
# Checking for persons with diabetes (outcome = 1)

Positive = data[data['Outcome']==1]
Positive.head(5)
```

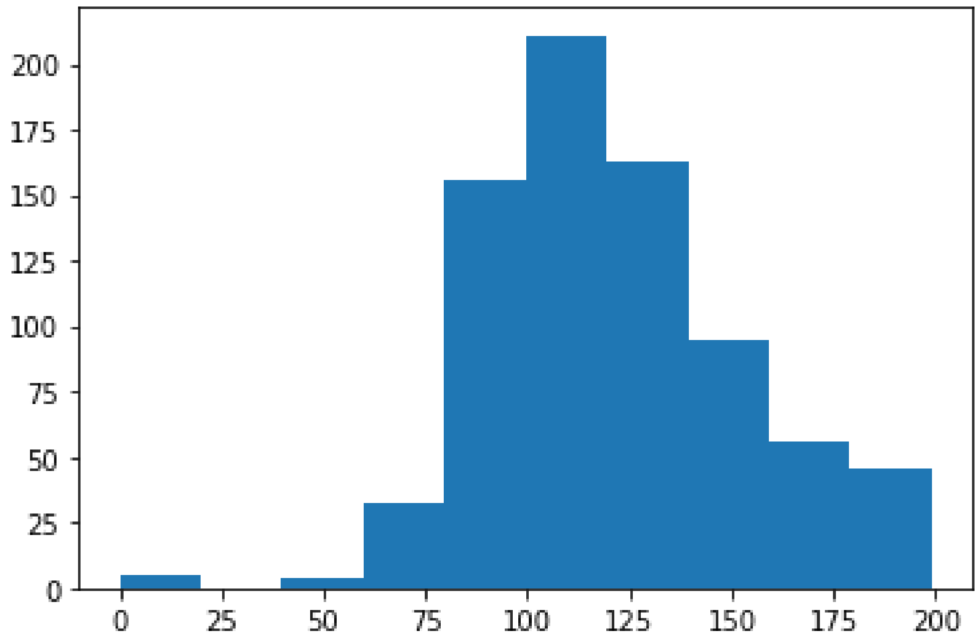
[7]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
2            8      183             64              0        0  23.3
4            0      137             40             35      168  43.1
6            3       78             50             32       88  31.0
8            2      197             70             45      543  30.5

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
2                     0.672   32        1
4                     2.288   33        1
6                     0.248   26        1
8                     0.158   53        1
```

[8]:
```
# Glucose levels

data['Glucose'].value_counts().head(7)
```

[8]:
```
100    17
99     17
129    14
125    14
111    14
106    14
95     13
Name: Glucose, dtype: int64
```

[9]:
```
# Visualization 1

plt.hist(data['Glucose'])
```

[9]:
```
(array([  5.,   0.,   4.,  32., 156., 211., 163.,  95.,  56.,  46.]),
 array([  0. ,  19.9,  39.8,  59.7,  79.6,  99.5, 119.4, 139.3, 159.2,
        179.1, 199. ]),
 <BarContainer object of 10 artists>)
```

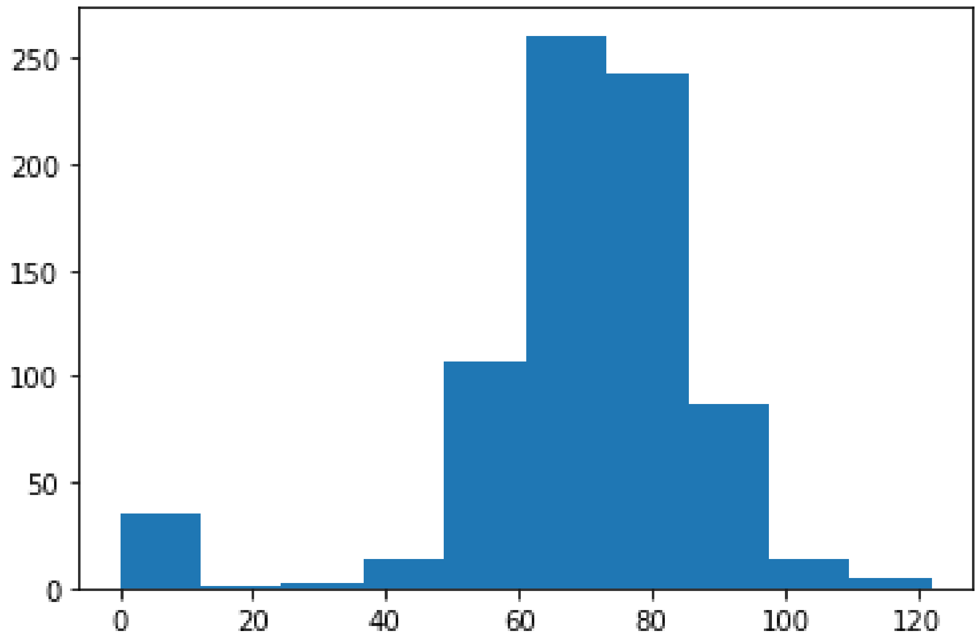


[10]:
```
# Blood Pressure

data['BloodPressure'].value_counts().head(7)
```

[10]:
```
70    57
74    52
68    45
78    45
72    44
64    43
80    40
Name: BloodPressure, dtype: int64
```

[11]:
```
# Visualization 2

plt.hist(data['BloodPressure'])
```

[11]:
```
(array([ 35.,   1.,   2.,  13., 107., 261., 243.,  87.,  14.,   5.]),
 array([  0. ,  12.2,  24.4,  36.6,  48.8,  61. ,  73.2,  85.4,  97.6,
        109.8, 122. ]),
 <BarContainer object of 10 artists>)
```

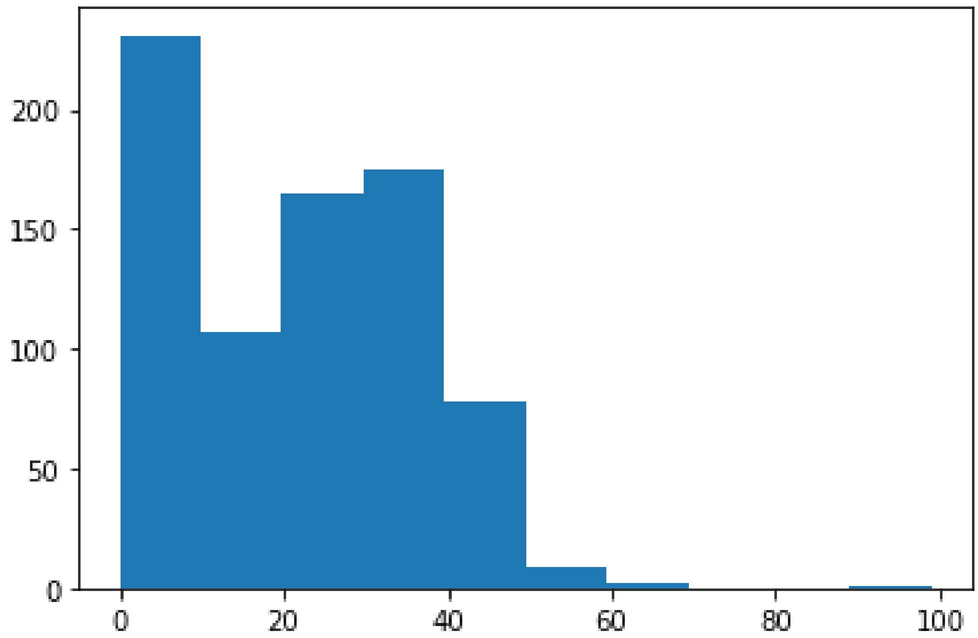


[12]:
```
# SkinThickness

data['SkinThickness'].value_counts().head(7)
```

[12]:
```
0     227
32     31
30     27
27     23
23     22
33     20
18     20
Name: SkinThickness, dtype: int64
```

[13]:
```
# Visulization 3

plt.hist(data['SkinThickness'])
```

[13]:
```
(array([231., 107., 165., 175.,  78.,   9.,   2.,   0.,   0.,   1.]),
 array([ 0. ,  9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99. ]),
 <BarContainer object of 10 artists>)
```

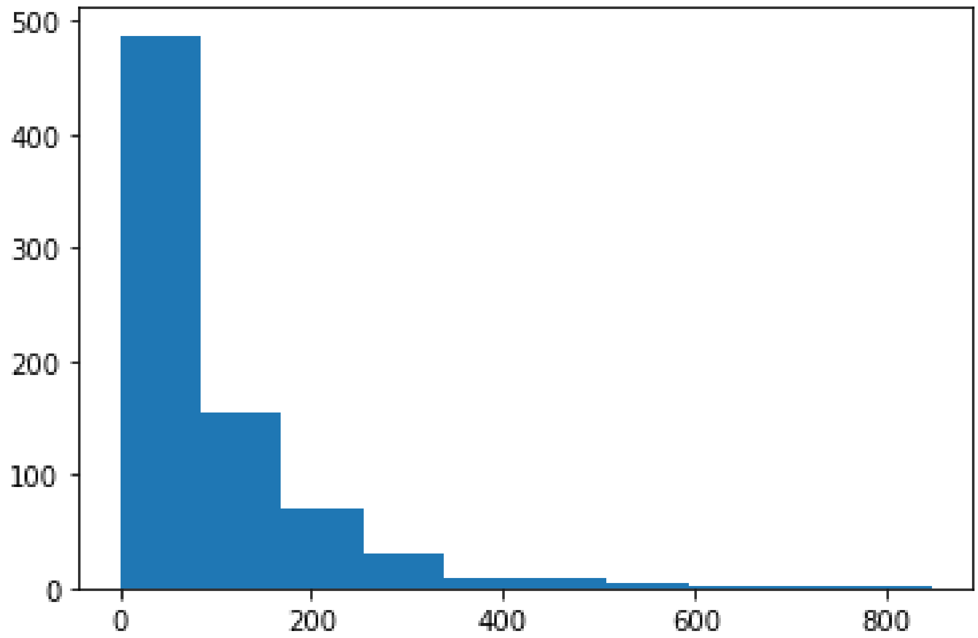


[14]:
```
# Insulin

data['Insulin'].value_counts().head(7)
```

[14]:
```
0      374
105     11
140      9
130      9
120      8
100      7
94       7
Name: Insulin, dtype: int64
```

[15]:
```
# Visualization 4

plt.hist(data['Insulin'])
```

[15]:
```
(array([487., 155.,  70.,  30.,   8.,   9.,   5.,   1.,   2.,   1.]),
 array([  0. ,  84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,
        761.4, 846. ]),
 <BarContainer object of 10 artists>)
```

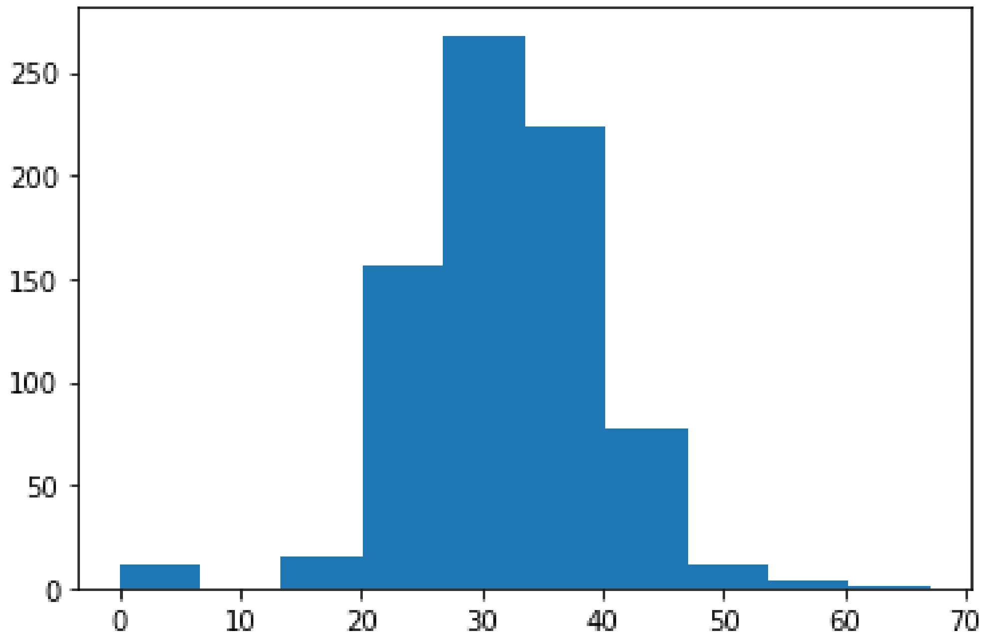


[16]:
```
# BMI values

data['BMI'].value_counts().head(7)
```

[16]:
```
32.0    13
31.6    12
31.2    12
0.0     11
33.3    10
32.4    10
32.8     9
Name: BMI, dtype: int64
```

[17]:
```
# Visualization 5

plt.hist(data['BMI'])
```

[17]:
```
(array([ 11.,   0.,  15., 156., 268., 224.,  78.,  12.,   3.,   1.]),
 array([ 0.  ,  6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,
        60.39, 67.1 ]),
 <BarContainer object of 10 artists>)
```



[18]:
```
data.describe().transpose()
```

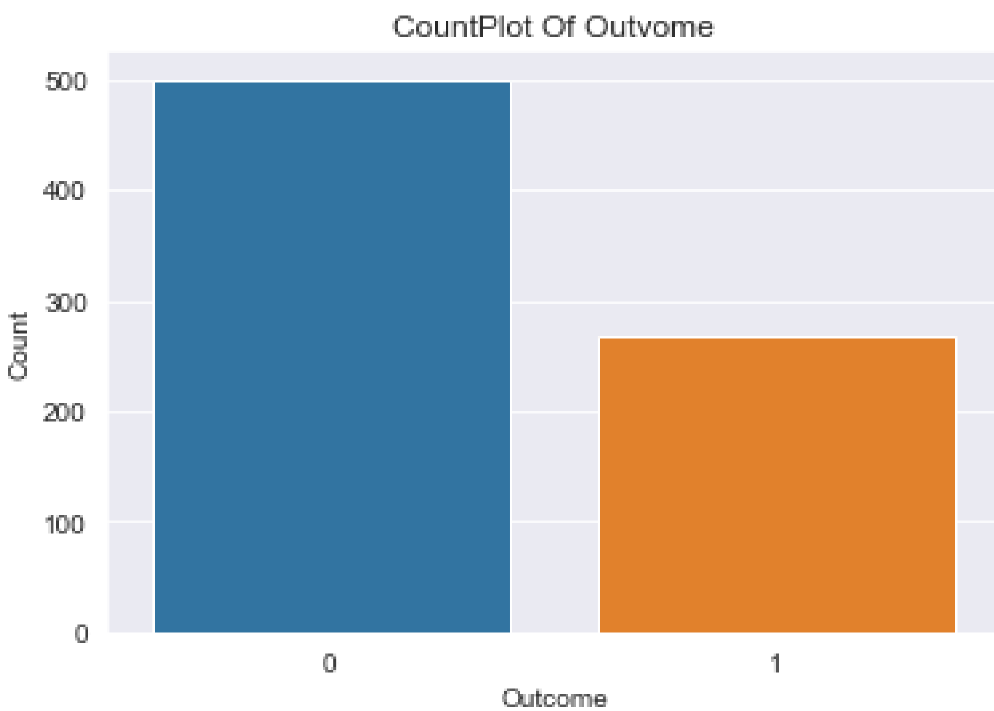
[18]:
```
                          count        mean         std     min       25%  \
Pregnancies               768.0    3.845052    3.369578   0.000   1.00000
Glucose                   768.0  120.894531   31.972618   0.000  99.00000
BloodPressure             768.0   69.105469   19.355807   0.000  62.00000
SkinThickness             768.0   20.536458   15.952218   0.000   0.00000
Insulin                   768.0   79.799479  115.244002   0.000   0.00000
BMI                       768.0   31.992578    7.884160   0.000  27.30000
DiabetesPedigreeFunction  768.0    0.471876    0.331329   0.078   0.24375
Age                       768.0   33.240885   11.760232  21.000  24.00000
Outcome                   768.0    0.348958    0.476951   0.000   0.00000

                               50%        75%     max
Pregnancies                 3.0000    6.00000   17.00
Glucose                   117.0000  140.25000  199.00
BloodPressure              72.0000   80.00000  122.00
SkinThickness              23.0000   32.00000   99.00
Insulin                    30.5000  127.25000  846.00
BMI                        32.0000   36.60000   67.10
DiabetesPedigreeFunction    0.3725    0.62625    2.42
Age                        29.0000   41.00000   81.00
Outcome                     0.0000    1.00000    1.00
```

[ ]:

2 *Week 2*

[19]:
```
# Visualization 6

# CountPlot

sns.set_style('darkgrid')
sns.countplot(data['Outcome'])
plt.title('CountPlot Of Outvome')
plt.xlabel('Outcome')
plt.ylabel('Count')
```

[19]:
```
Text(0, 0.5, 'Count')
```



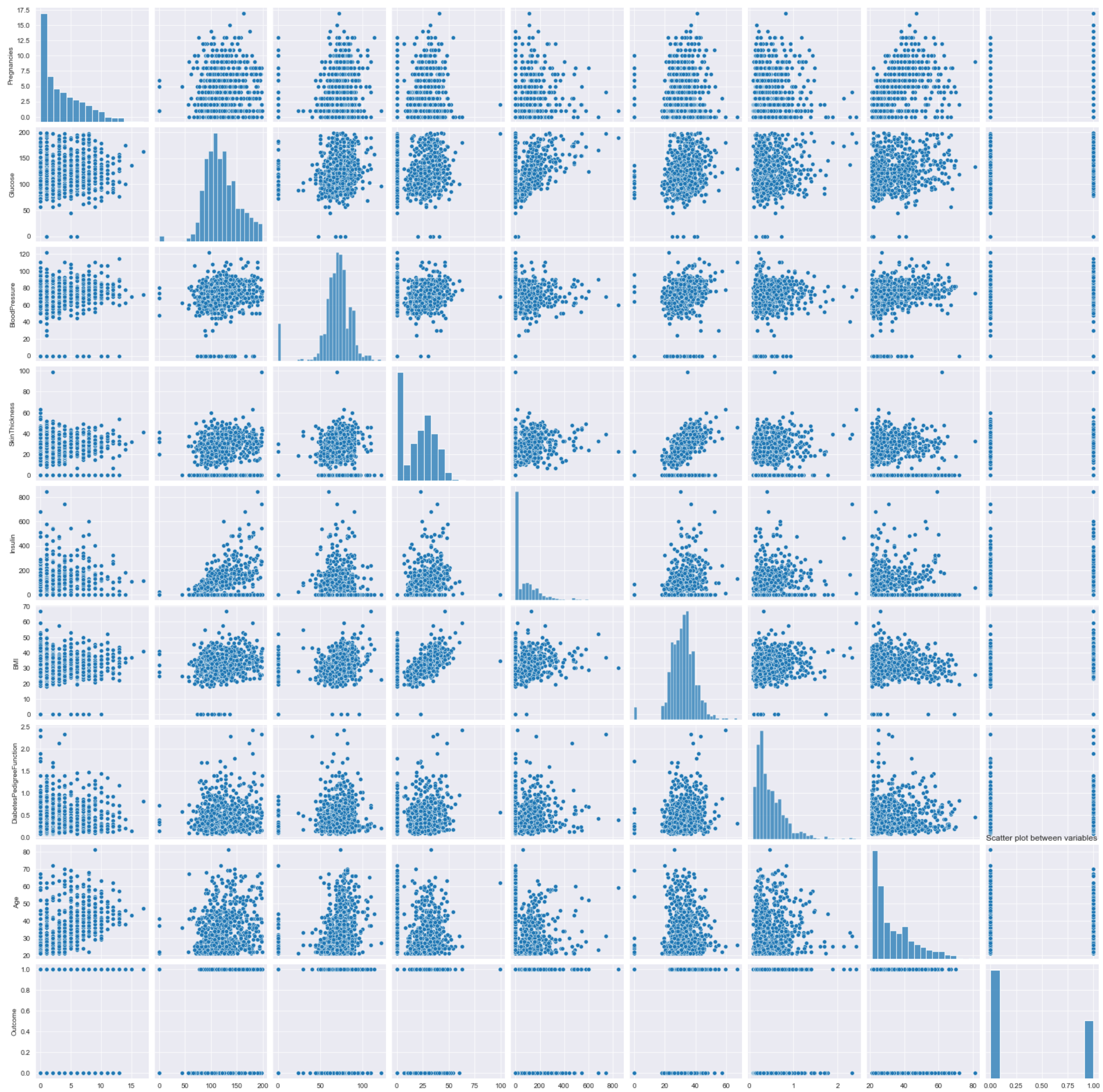
[20]:
```
# Count of Variables

print('Count of class is:\n',data['Outcome'].value_counts())
```

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

No need of Sampling as the data was balanced. We can move forward to training and testing procedure. But there is medical data which we need to make sure of Type 2 Error by using a ROC curve

[21]:
```python
# Visualization 7

# Scatter Plot

sns.pairplot(data)
plt.title('Scatter plot between variables')
```

[21]: 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.

[22]:
```
# Correlation Analysis

data.corr()
```

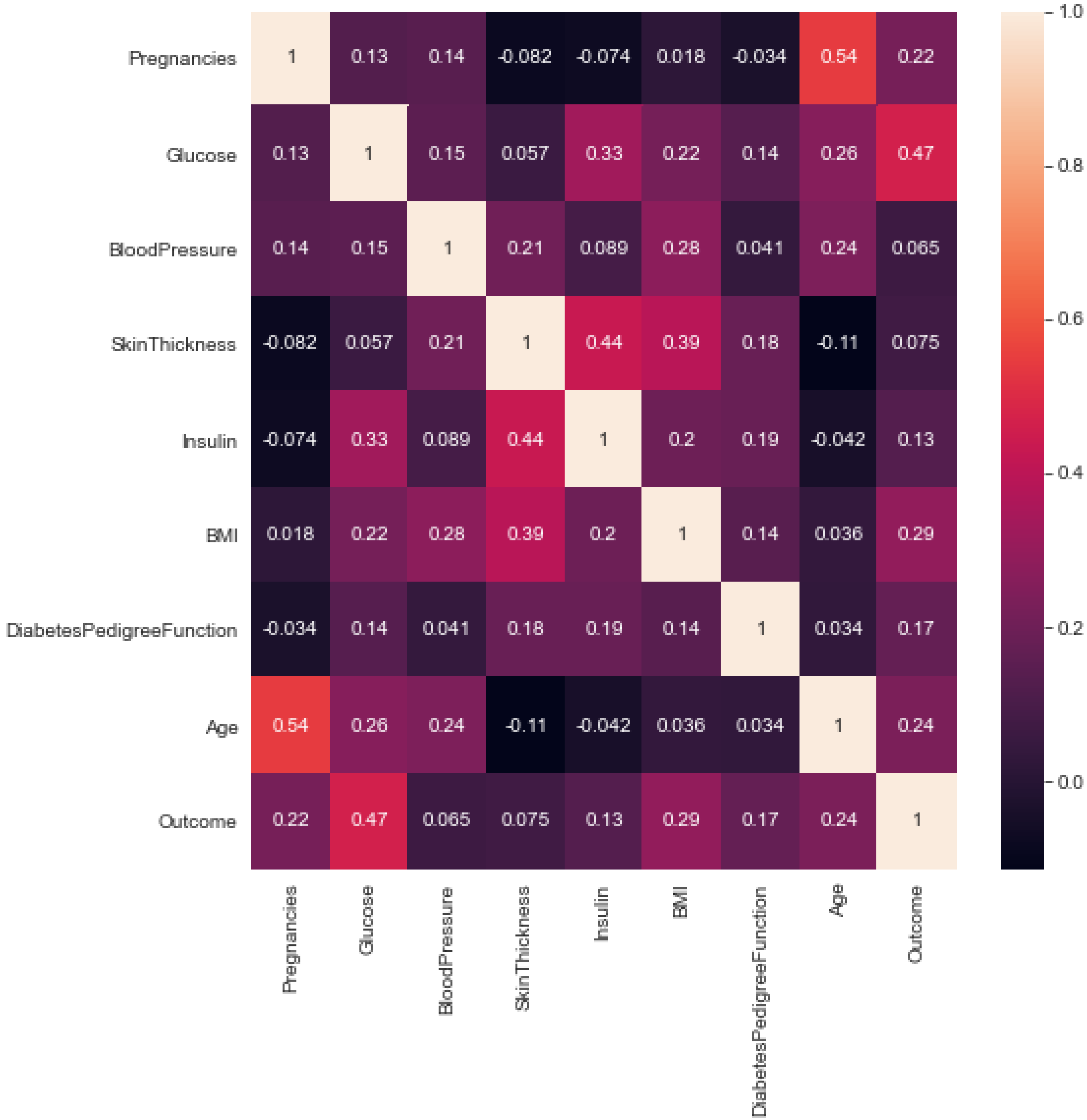
[22]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

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.

[25]:
```
# Visualization 8

# Correlation Values

plt.subplots(figsize=(8,8))
sns.heatmap(data.corr(),annot=True)
```

[25]: <AxesSubplot:>



3 *Week3*

[26]:
```
data.head()
```

[26]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	

```
3              1       89              66             23        94  28.1
4              0      137              40             35       168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

4 Data Preprocessing

[27]:
```
x=data.iloc[:,:-1].values
y=data.iloc[:,-1].values
```

[28]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

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

[30]:
```
Scale = StandardScaler()
X_train_std = Scale.fit_transform(X_train)
X_test_std = Scale.transform(X_test)
```

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

[32]:
```
data_norm = data.iloc[:,:-1]
```

[33]:
```
data_normalized = data_norm.apply(Norm)
```

[34]:
```
X_train_norm,X_test_norm,y_train_norm,y_test_norm=train_test_split(data_normalized.
 ↪values,y,test_size=0.20,random_state=0)
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 here. 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.

4.0.1 KNN with standard scaling

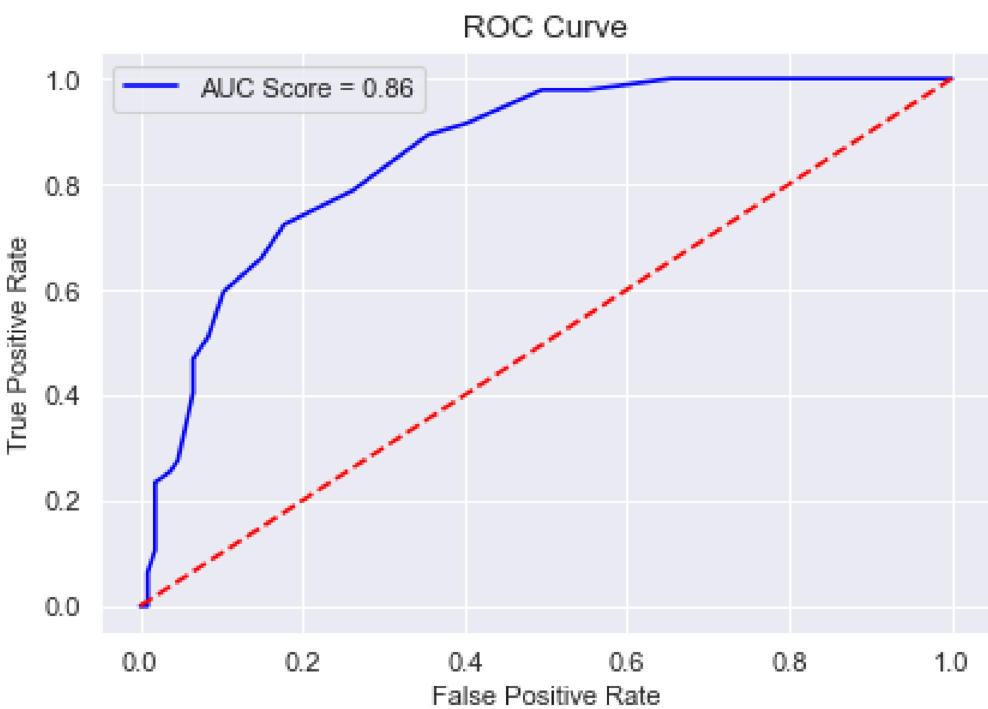
[35]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=25)
#Using 25 neighbors just as thumb rule sqrt of observation
knn_model.fit(X_train_std,y_train)
knn_pred=knn_model.predict(X_test_std)
```

[36]:
```
# Visualization 9

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::
```

```
0.7922077922077922

 Classification Report::
               precision    recall  f1-score   support

           0       0.81      0.92      0.86       107
           1       0.73      0.51      0.60        47

    accuracy                           0.79       154
   macro avg       0.77      0.71      0.73       154
weighted avg       0.78      0.79      0.78       154



 ROC Curve
```

[36]: <matplotlib.legend.Legend at 0x267c99e6b80>



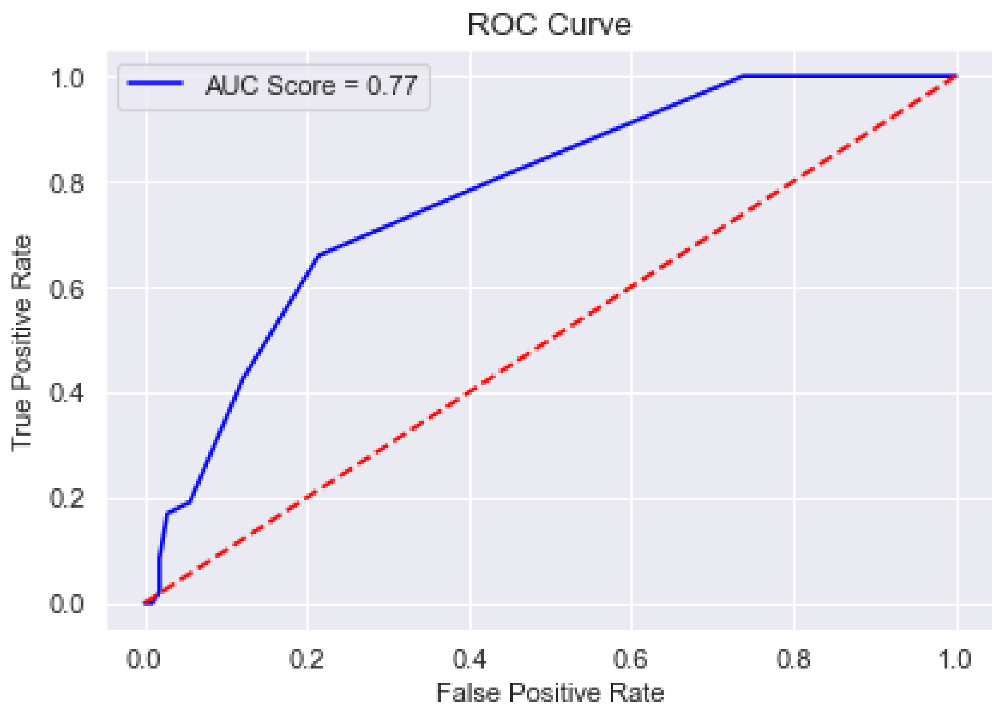
4.0.2 KNN with Normalization

[37]:
```
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)
```

[38]:
```
# Visualization 10

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::
0.7922077922077922

 Classification Report::
              precision    recall  f1-score   support

           0       0.82      0.91      0.86       107
           1       0.71      0.53      0.61        47

    accuracy                           0.79       154
   macro avg       0.76      0.72      0.73       154
weighted avg       0.78      0.79      0.78       154



 ROC Curve
```

[38]: <matplotlib.legend.Legend at 0x267c9a65730>



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.

4.1 Support Vector Classifier

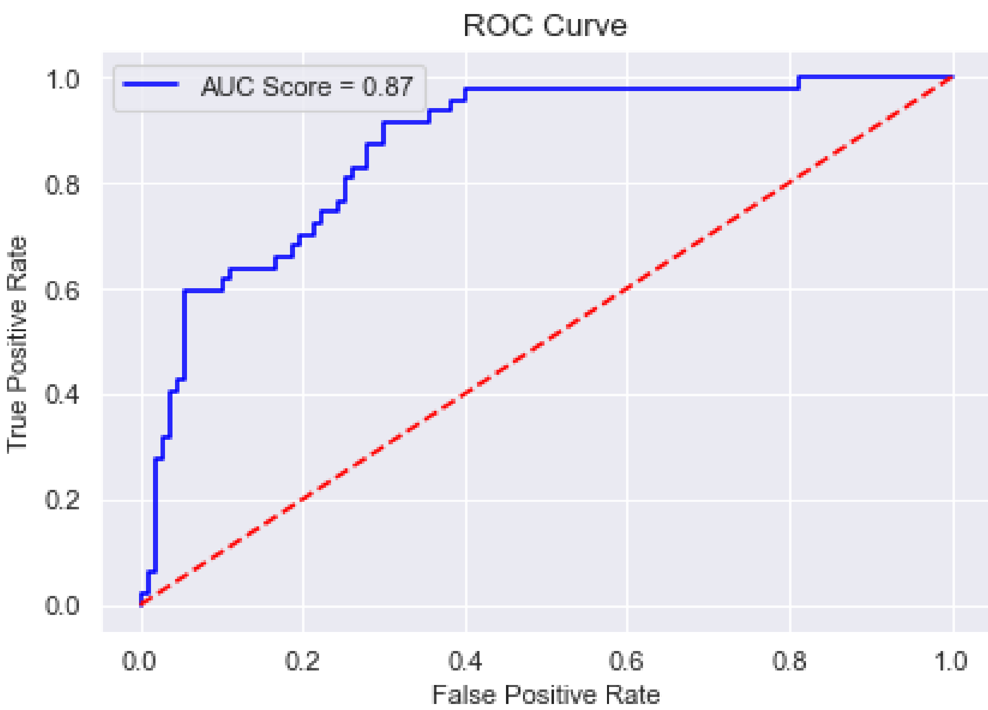
[39]:
```
from sklearn.svm import SVC
svc_model_linear = SVC(kernel='linear',random_state=0,probability=True,C=0.01)
svc_model_linear.fit(X_train_std,y_train)
svc_pred=svc_model_linear.predict(X_test_std)
```

[40]:
```
# Visualization 11

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::
0.8246753246753247

 Classification Report::
               precision    recall  f1-score   support

           0       0.84      0.93      0.88       107
           1       0.78      0.60      0.67        47

    accuracy                           0.82       154
   macro avg       0.81      0.76      0.78       154
weighted avg       0.82      0.82      0.82       154



 ROC Curve
```

[40]: `<matplotlib.legend.Legend at 0x267c9ad5880>`

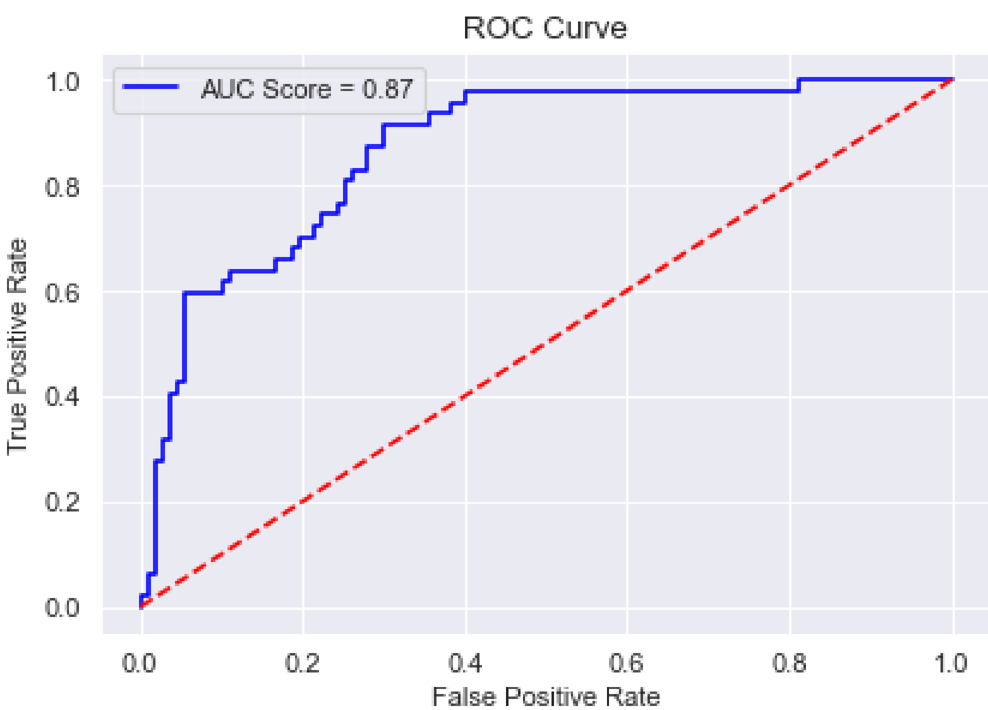


[41]:
```
from sklearn.svm import SVC
svc_model_rbf = SVC(kernel='rbf',random_state=0, probability=True,C=1)
svc_model_rbf.fit(X_train_std, y_train)
svc_pred_rbf=svc_model_rbf.predict(X_test_std)
```

[42]:
```
# Visualization 12

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::
0.7922077922077922

 Classification Report::
               precision    recall  f1-score   support

           0       0.82      0.90      0.86       107
           1       0.70      0.55      0.62        47

    accuracy                           0.79       154
   macro avg       0.76      0.73      0.74       154
weighted avg       0.78      0.79      0.78       154



 ROC Curve
```

[42]: <matplotlib.legend.Legend at 0x267c9b28460>



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.

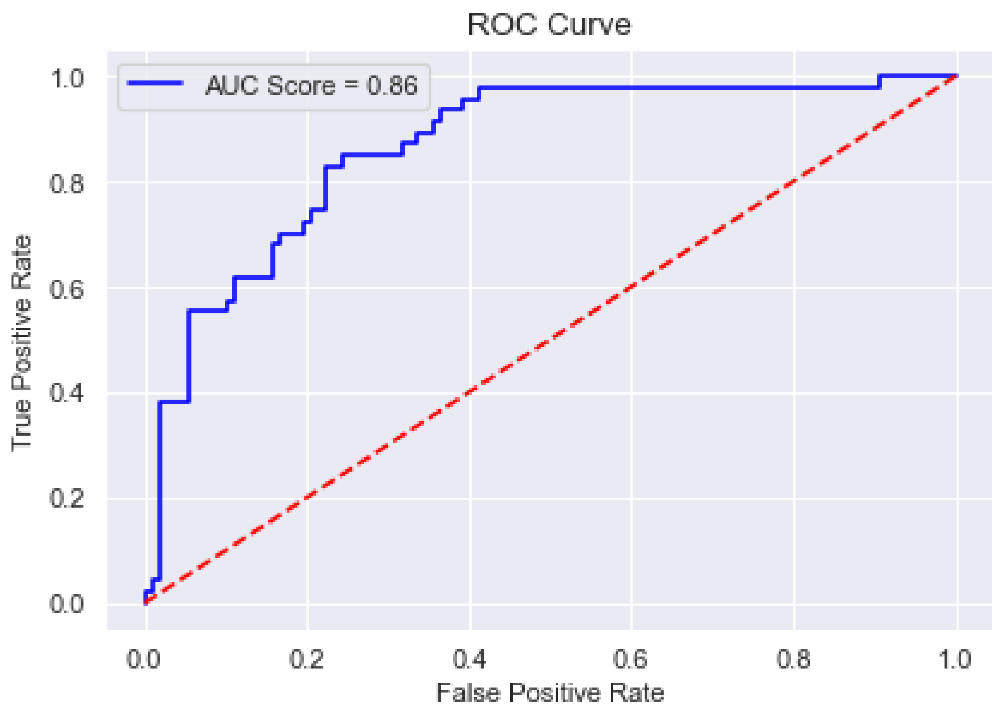
4.1.1 Logistic Regression

[45]:
```
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)
```

[46]:
```
# Visualization 13

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::
0.7987012987012987

 Classification Report::
              precision    recall  f1-score   support

           0       0.80      0.94      0.87       107
           1       0.79      0.47      0.59        47

    accuracy                           0.80       154
   macro avg       0.79      0.71      0.73       154
weighted avg       0.80      0.80      0.78       154



 ROC Curve
```

[46]: <matplotlib.legend.Legend at 0x267c9b95250>



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

4.1.2 Ensemble Learning - Random Forest

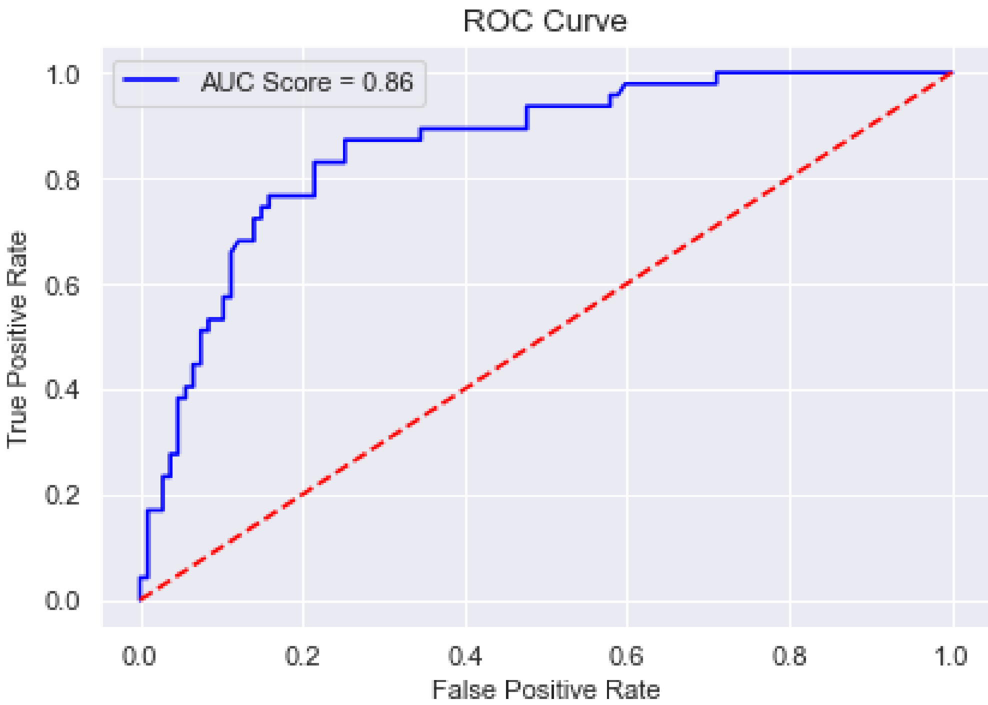
[47]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=1000, random_state=0)
rf_model.fit(X_train_std,y_train)
rf_pred = rf_model.predict(X_test_std)
```

[48]:
```
#Visualization 14

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::
0.8181818181818182

 Classification Report::
               precision    recall  f1-score   support

           0       0.86      0.89      0.87       107
           1       0.72      0.66      0.69        47

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



 ROC Curve
```

[48]: <matplotlib.legend.Legend at 0x267cad72070>



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

```
[ ]:
```