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

In [2]:
```python
data = pd.read_csv(r"D:\family\eswar\simplilearn\capstone\health care\Project 2\Healthcare - Diabetes\health ca
```

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

Out[3]:

	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 [4]:
```python
data.isnull().any()
```

Out[4]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

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

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

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

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	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	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [7]:
```python
Positive = data[data['Outcome']==1]
Positive.head(5)
```

Out[7]:

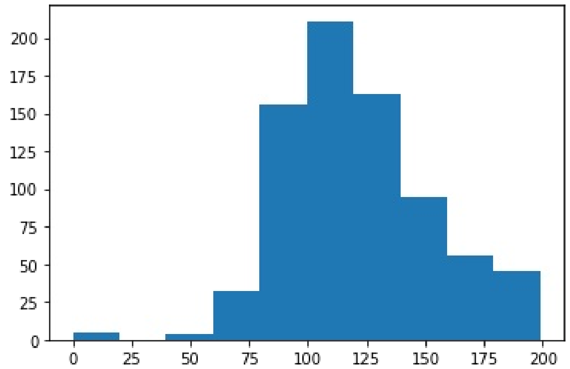
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
2	8	183	64	0	0	23.3	0.672	32	1
4	0	137	40	35	168	43.1	2.288	33	1
6	3	78	50	32	88	31.0	0.248	26	1
8	2	197	70	45	543	30.5	0.158	53	1

In [8]:
```
data['Glucose'].value_counts().head(7)
```

Out[8]:
```
99     17
100    17
111    14
129    14
125    14
106    14
112    13
Name: Glucose, dtype: int64
```

In [9]:
```
plt.hist(data['Glucose'])
```

Out[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>)
```

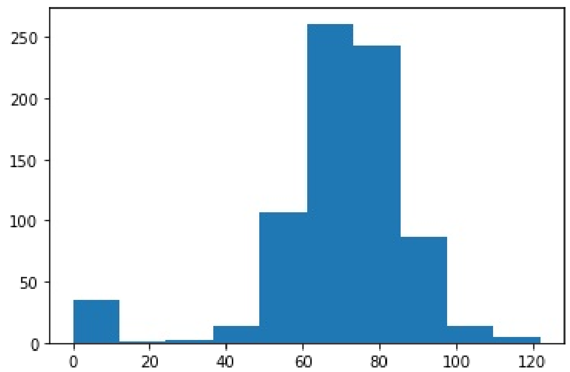


In [10]:
```
data['BloodPressure'].value_counts().head(7)
```

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

In [11]:
```
plt.hist(data['BloodPressure'])
```

Out[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>)
```



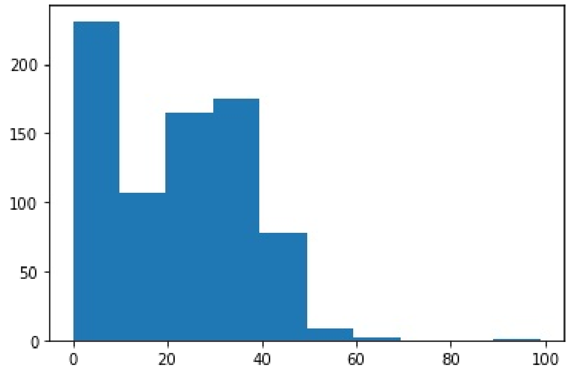
In [12]:
```
data['SkinThickness'].value_counts().head(7)
```

Out[12]:
```
0     227
32     31
30     27
27     23
23     22
33     20
28     20
Name: SkinThickness, dtype: int64
```

In [13]:
```
plt.hist(data['SkinThickness'])
```

In [13]:
```
plt.hist(data['SkinThickness'])
```

Out[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>)
```

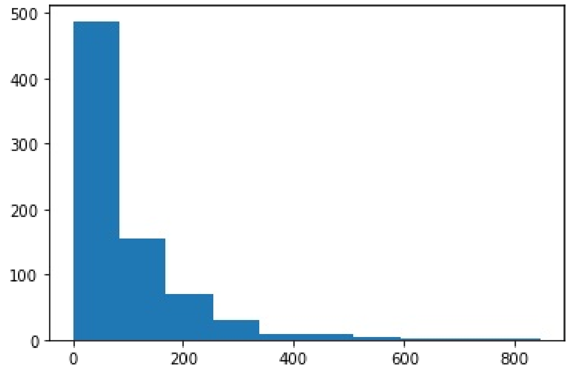


In [14]:
```
data['Insulin'].value_counts().head(7)
```

Out[14]:
```
0      374
105     11
130      9
140      9
120      8
94       7
180      7
Name: Insulin, dtype: int64
```

In [15]:
```
plt.hist(data['Insulin'])
```

Out[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>)
```

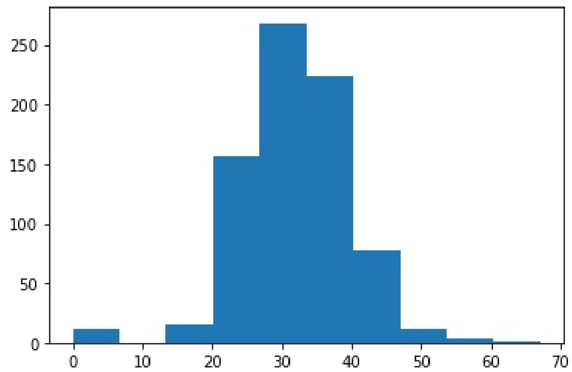


In [16]:
```
data['BMI'].value_counts().head(7)
```

Out[16]:
```
32.0    13
31.6    12
31.2    12
0.0     11
32.4    10
33.3    10
30.1     9
Name: BMI, dtype: int64
```

In [17]:
```
plt.hist(data['BMI'])
```

Out[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>)
```



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

Out[18]:

	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
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 [19]:

```
data.head()
```

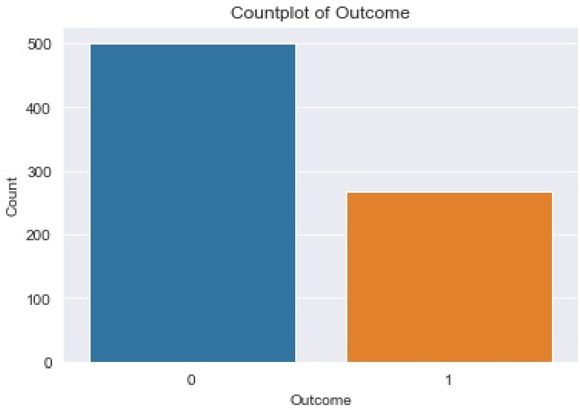
Out[19]:

	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 [20]:

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

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



In [21]:

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

Out[21]:

```
Text(0.5, 1.0, 'scatter plot between variables')
```

In [22]:
```
data.corr()
```

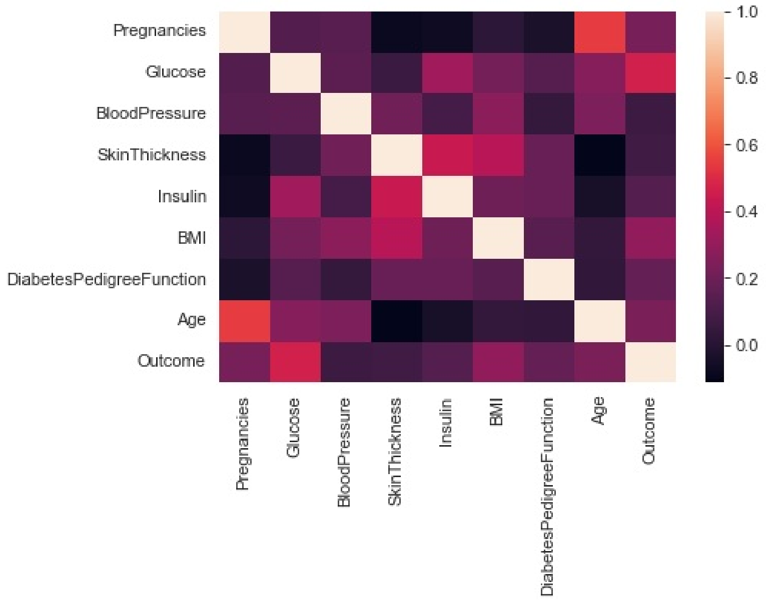
Out[22]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	O
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.

In [23]:

```
plt.figure(dpi=80)
#create correlation heat map
sns.heatmap(data.corr())
```

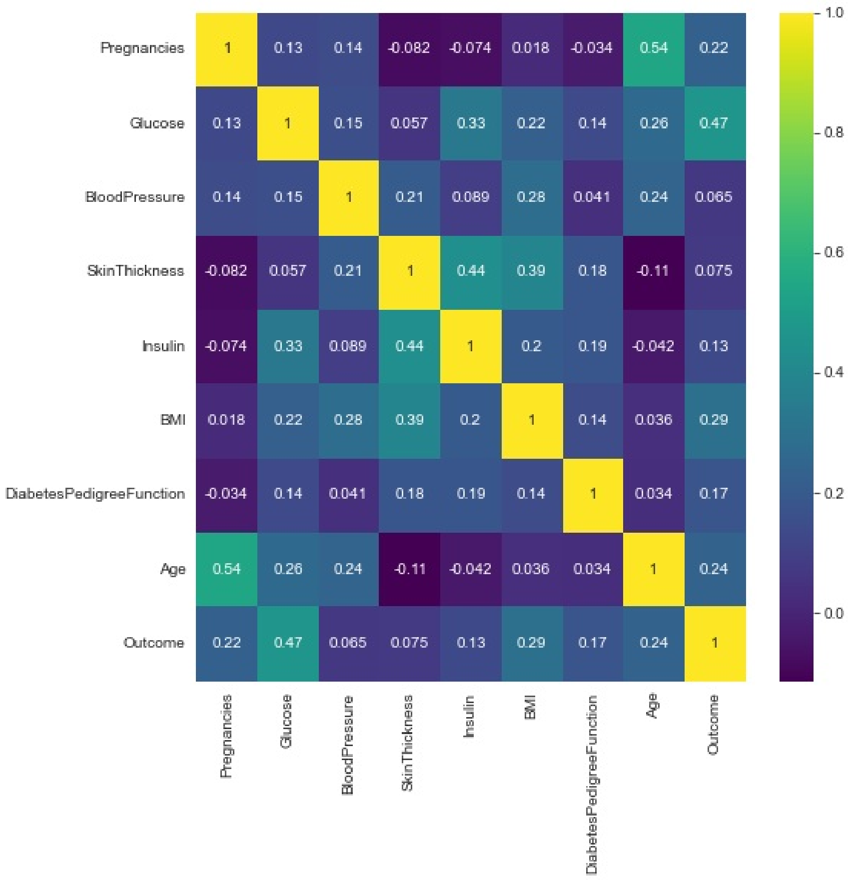
Out[23]: <AxesSubplot:>



In [24]:

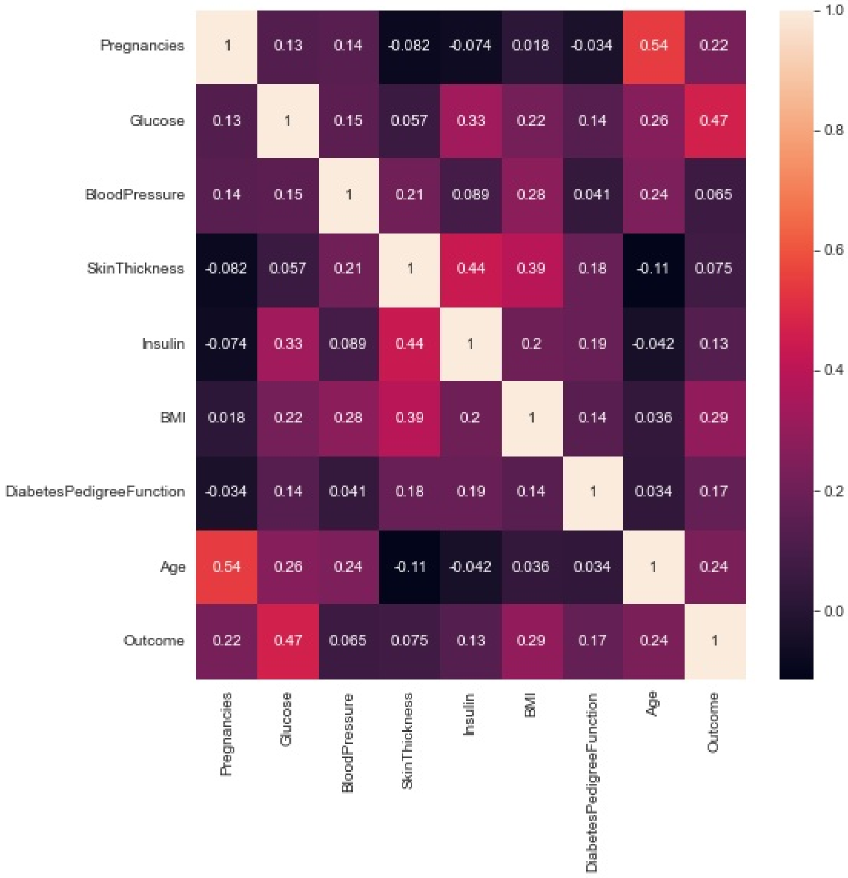
```
plt.subplots(figsize=(8,8))
sns.heatmap(data.corr(),annot=True,cmap='viridis')  ### gives correlation value
```

Out[24]: <AxesSubplot:>



In [25]:
```
plt.subplots(figsize=(8,8))
sns.heatmap(data.corr(),annot=True)  ### gives correlation value
```

Out[25]: <AxesSubplot:>



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

Out[26]:

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

In [28]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split(x, y, test_size=0.20,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,)
```

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

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

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

```
In [34]: x_train_norm,x_test_norm,y_train_norm,y_test_norm=train_test_split(data_normalized.values,y,test_size=0.20,rand
         print(x_train_norm.shape)
         print(x_test_norm.shape)
         print(y_train_norm.shape)
         print(y_test_norm.shape)
```

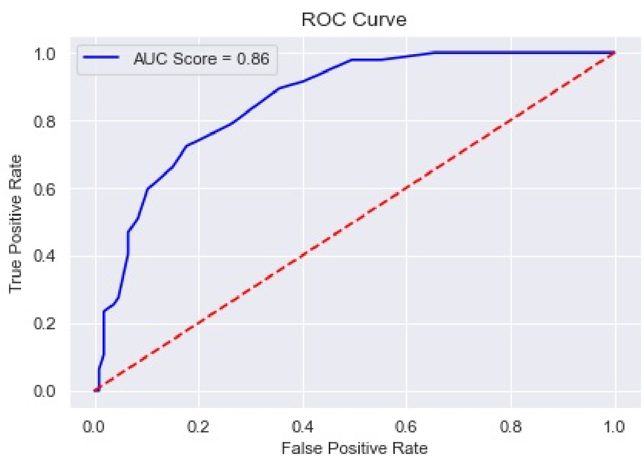
```
(614, 8)
(154, 8)
(614,)
(154,)
```

```
In [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)
```

```
In [36]: 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
```

```
Out[36]: <matplotlib.legend.Legend at 0x27c47d5fe20>
```




```
In [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)
```

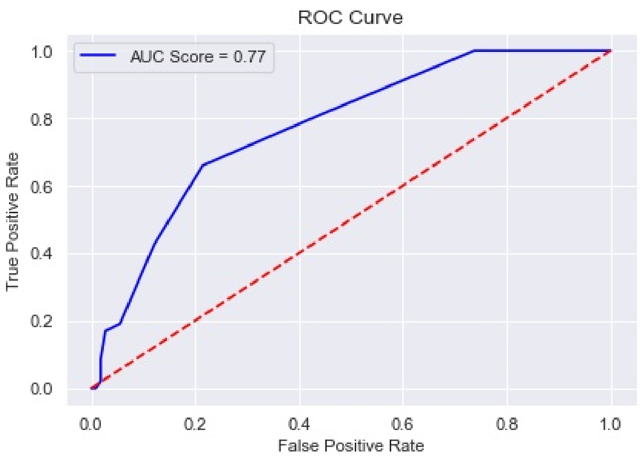
```
In [38]: print("Model Validation ==>\n")
         print("Accuracy Score of KNN Model with Normalization::")
         print(metrics.accuracy_score(y_test_norm,knn_pred_norm))
```

```
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
```

Out[38]:

```
<matplotlib.legend.Legend at 0x27c47dae4f0>
```



In [39]:

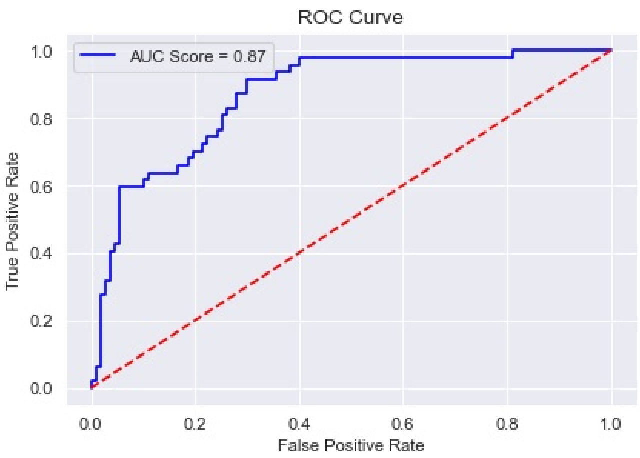
```
from sklearn.svm import SVC
svc_model_linear = SVC(kernel='linear',random_state=0,probability=True,C=0.01)
svc_model_linear.fit(x_train_std,y_train)
svc_pred=svc_model_linear.predict(x_test_std)
```

In [40]:

```
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
```

Out[40]: `<matplotlib.legend.Legend at 0x27c47e1be80>`

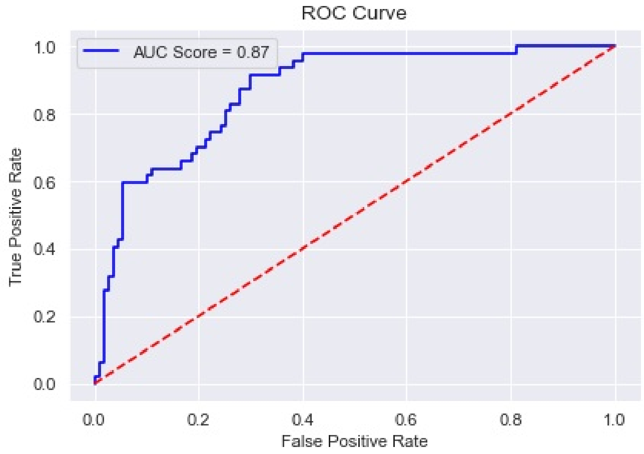


In [41]:
```
from sklearn.svm import SVC
svc_model_rbf = SVC(kernel='rbf',random_state=0, probability=True,C=1)
svc_model_rbf.fit(x_train_std, y_train)
svc_pred_rbf=svc_model_rbf.predict(x_test_std)
```

In [42]:
```
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
```

Out[42]: `<matplotlib.legend.Legend at 0x27c47e79fd0>`

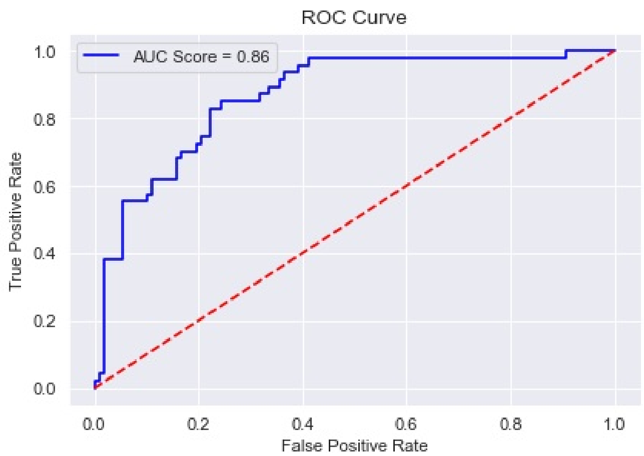


In [43]:
```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(C=0.01)
lr_model.fit(x_train_std, y_train)
lr_pred=lr_model.predict(x_test_std)
```

In [44]:
```
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
```

Out[44]:
```
<matplotlib.legend.Legend at 0x27c47ee4c70>
```



In [45]:
```python
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 [46]:
```python
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
```

Out[46]:
```
<matplotlib.legend.Legend at 0x27c481b0070>
```