In [40]:
```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import seaborn as sns
import matplotlib as plt
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```python
df = pd.read_csv('/Users/ranood/Desktop/Project 2/HealthcareDiabetes/healthCareDiabetes.csv')
```

In [3]:
```python
df
```

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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [4]:
```python
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 [5]:
```python
df.describe()
```

Out[5]:

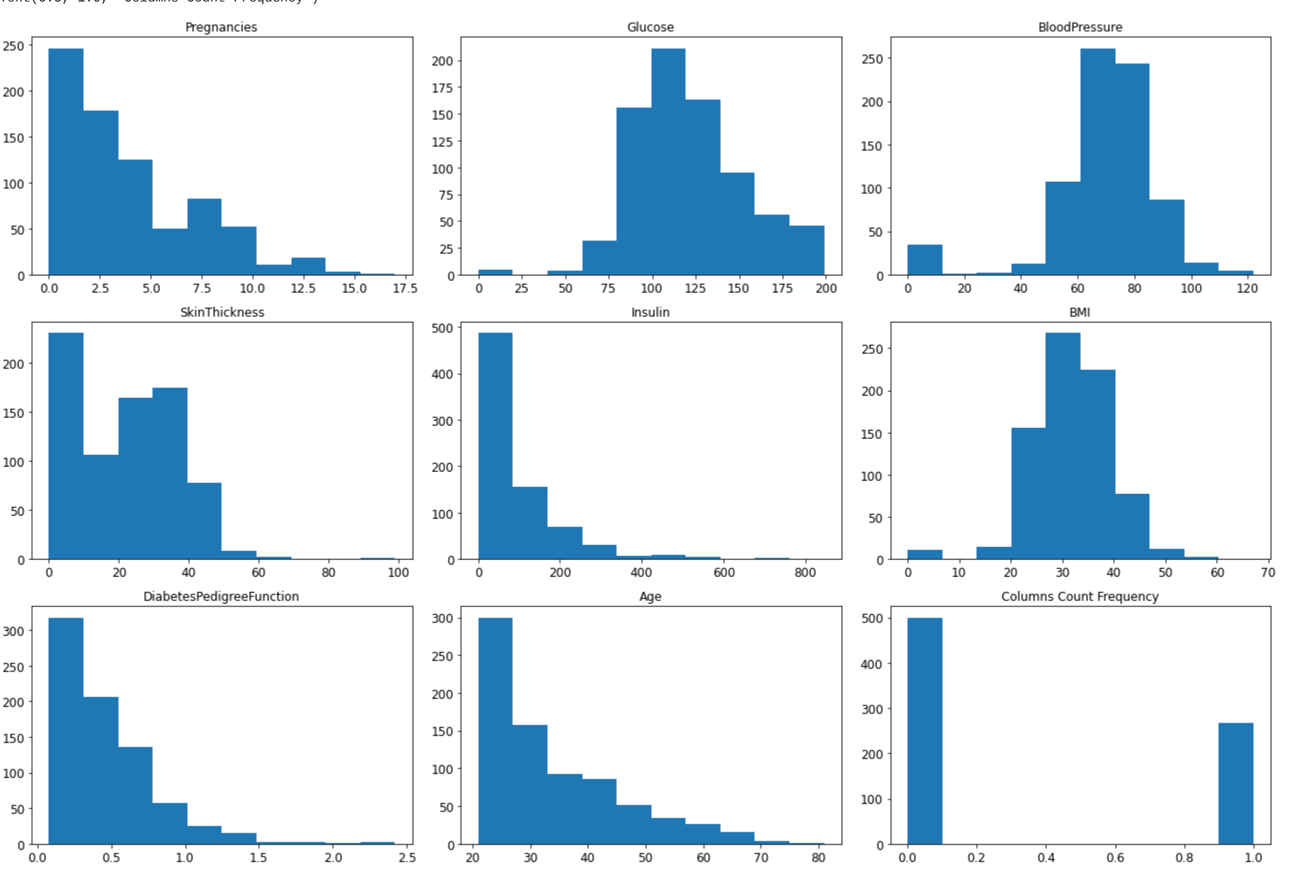
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	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 [6]:
```python
df['Outcome'].value_counts()
```

Out[6]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [7]:
```python
#plot each column with its frequency
df.hist(
        xlabelsize = 12,
        ylabelsize = 12,
        linewidth = 3.0,
        grid =False )
plt.tight_layout(rect=(0, 0, 3, 3))# change the size
plt.title('Columns Count Frequency')
```

Out[7]:
```
Text(0.5, 1.0, 'Columns Count Frequency')
```



In [8]:
```python
(df==0).sum(axis=0)
```

Out[8]:
```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

In [9]:
```python
features = pd.DataFrame(df)
features.drop('Outcome', axis =1, inplace =True)
#replace 0 (missing) values with the column mean
for i in features.columns:
    features[i] = features[i].replace(0,features[i].mean())
```

In [10]:
```python
features.head()
```

Out[10]:

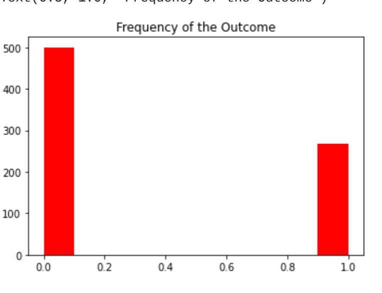
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6.000000	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1.000000	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8.000000	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1.000000	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	3.845052	137.0	40.0	35.000000	168.000000	43.1	2.288	33

In [11]:
```python
(features==0).sum(axis =0)
```

Out[11]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
dtype: int64
```

In [12]:
```python
df['Outcome'].hist(grid = False, color = 'red')
plt.title('Frequency of the Outcome')
```

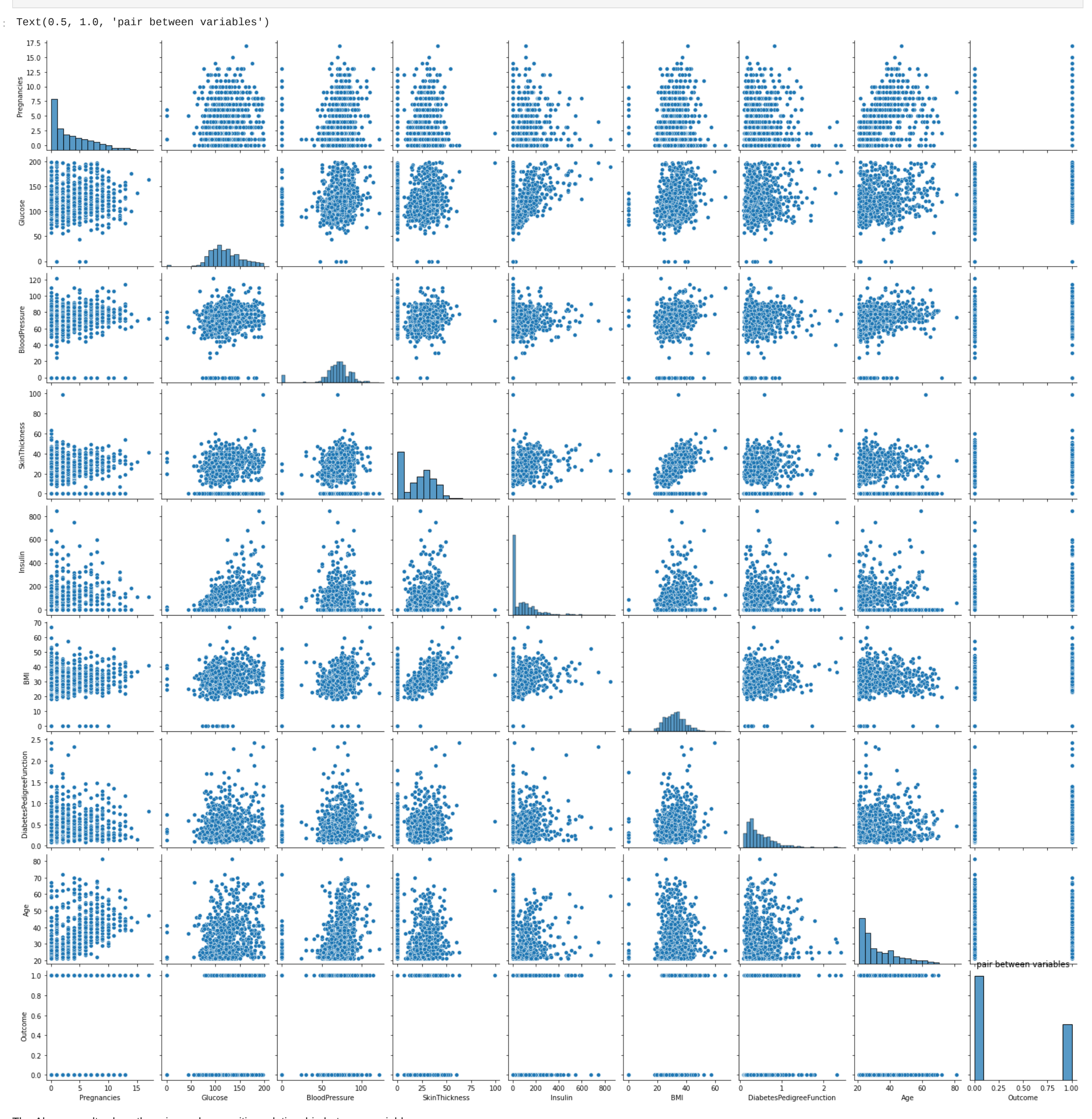
Out[12]:
```
Text(0.5, 1.0, 'Frequency of the Outcome')
```



The Above result shows that 500 people out of 700 have no diabetes and the rest have

In [13]:
```python
sns.pairplot(df)
plt.title('pair between variables')
```

Out[13]:
```
Text(0.5, 1.0, 'pair between variables')
```



The Above results show there is no clear positive relationship between variables.

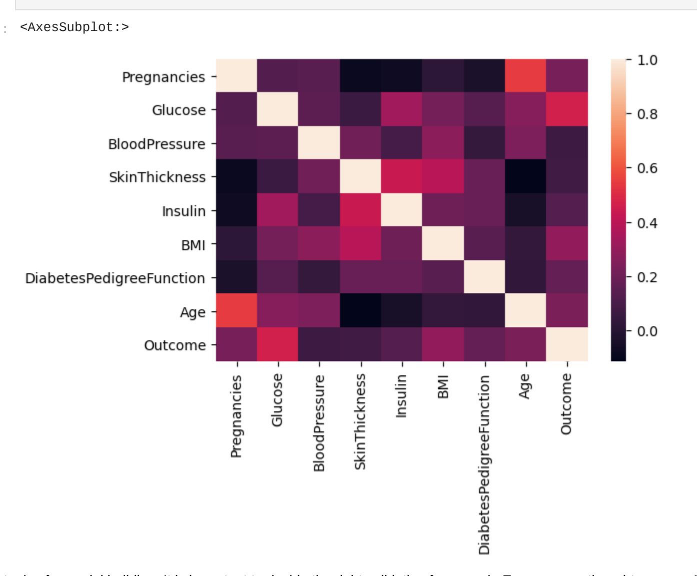
In [14]:
```python
df.corr()
```

Out[14]:

	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

In [15]:
```python
#plot correlation matrix
plt.figure(dpi=100)
sns.heatmap(df.corr())
```

Out[15]:
```
<AxesSubplot:>
```



Devise strategies for model building. It is important to decide the right validation framework. Express your thought process 1- scale and normalize the data 2- split the data 3- create the model (could be logistic regression) 4- make the prediction 5- measure the accuracy by accuracy metrics and confusion matrix

In [16]:
```python
#Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm

X = df.iloc[:,:-1]
y = df.iloc[:,-1]

#scaling the data
scale=StandardScaler()

X=scale.fit_transform(X)

#split the data to training and testing
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state =0)
```

In [17]:
```python
X_train.shape
```

Out[17]:
```
(614, 8)
```

In [18]:
```python
X_test.shape
```

Out[18]:
```
(154, 8)
```

In [19]:
```python
y_train.shape
```

Out[19]:
```
(614,)
```

In [20]:
```python
y_test.shape
```

Out[20]:
```
(154,)
```

In [21]:
```python
#normalize and split the data
n = lambda a:(a-min(a))/(max(a)-min(a))
X_norm = df.iloc[:,:-1]
X_norm = X_norm.apply(n)
X_train_norm,X_test_norm,y_train,y_test=train_test_split(X_norm,y,test_size=0.20,random_state =0)
```

In [22]:
```python
X_train_norm.shape
```

Out[22]:
```
(614, 8)
```

logistic regression model

In [23]:
```python
lr = LogisticRegression()
lr.fit(X_train,y_train)
```

Out[23]:
```
LogisticRegression()
```

In [24]:
```python
pred=lr.predict(X_test)
```

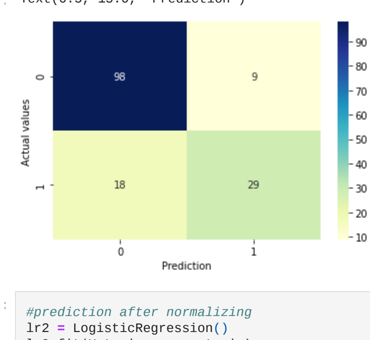
In [25]:
```python
metrics.accuracy_score(y_test, pred)
```

Out[25]:
```
0.8246753246753247
```

In [26]:
```python
cnf_matrix = metrics.confusion_matrix(y_test, pred)
```

In [27]:
```python
sns.heatmap(pd.DataFrame(cnf_matrix),annot=True, cmap="YlGnBu" ,fmt='g')
plt.ylabel('Actual values')
plt.xlabel('Prediction')
```

Out[27]:
```
Text(0.5, 15.0, 'Prediction')
```



In [33]:
```python
#prediction after normalizing
lr2 = LogisticRegression()
lr2.fit(X_train_norm,y_train)
pred2=lr.predict(X_test_norm)
metrics.accuracy_score(y_test, pred2)
```

Out[33]:
```
0.551948051948052
```

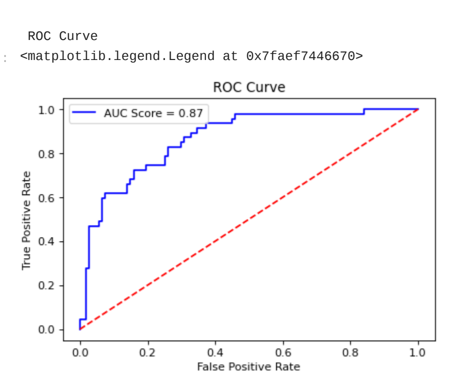
In [43]:
```python
fpr, tpr, thresholds = roc_curve(y_test, lr.predict_proba(X_test)[:,1])
```

In [44]:
```python
lr_roc = roc_auc_score(y_test, pred)
```

In [63]:
```python
print(metrics.accuracy_score(y_test,pred))
print("\n","Logistic Regression Report")
print(metrics.classification_report(y_test,pred),'\n')
print("\n","ROC Curve")
lr_prob=lr.predict_proba(X_test)
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()
```

```
0.8246753246753247

 Logistic Regression Report
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       107
           1       0.76      0.62      0.68        47

    accuracy                           0.82       154
   macro avg       0.80      0.77      0.78       154
weighted avg       0.82      0.82      0.82       154



 ROC Curve
```

Out[63]:
```
<matplotlib.legend.Legend at 0x7faef7446670>
```



KNN Model

In [29]:
```python
KNN = KNeighborsClassifier(20)
KNN.fit(X_train, y_train)
```

Out[29]:
```
KNeighborsClassifier(n_neighbors=20)
```

In [30]:
```python
y_pred = KNN.predict(X_test)
```

In [31]:
```python
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8051948051948052
```

In [57]:
```python
#prediction after normalizing
KNN2 = KNeighborsClassifier(20)
KNN2.fit(X_train_norm, y_train)
y_pred = KNN2.predict(X_test_norm)
metrics.accuracy_score(y_test, y_pred)
```

Out[57]:
```
0.8116883116883117
```

In [61]:
```python
print(metrics.classification_report(y_test,y_pred),'\n')
print("\n","ROC Curve")
knn_prob=KNN2.predict_proba(X_test)
knn_prob_norm1 = knn_prob[:,1]
fpr,tpr,thresh = metrics.roc_curve(y_test,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()
```

```
              precision    recall  f1-score   support

           0       0.82      0.93      0.87       107
           1       0.76      0.55      0.64        47

    accuracy                           0.81       154
   macro avg       0.79      0.74      0.76       154
weighted avg       0.81      0.81      0.80       154



 ROC Curve
```

Out[61]:
```
<matplotlib.legend.Legend at 0x7faef71ac850>
```


Logistic Regression Without Normalizing is the best model and then KNN after normalizing. Also, AUC for logistic regression is the best.

In [ ]:

In [ ]: