Healthcare_Capstone_Project_Darshana.N

June 1, 2020

0.1 Data Science Capstone Project

1 Project Title : Health Care

1.1 Week one Task(Health care) : Data Exploration:

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```python
df=pd.read_csv("health care diabetes.csv")
df.head()
```

[2]:
```
   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
```

Descriptive Analysis on Data

[3]:
```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
```

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

[4]:
```
       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
```

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

[5]:
```
print("standard deviation of each variables")
df.apply(np.std)
```

```
standard deviation of each variables
```

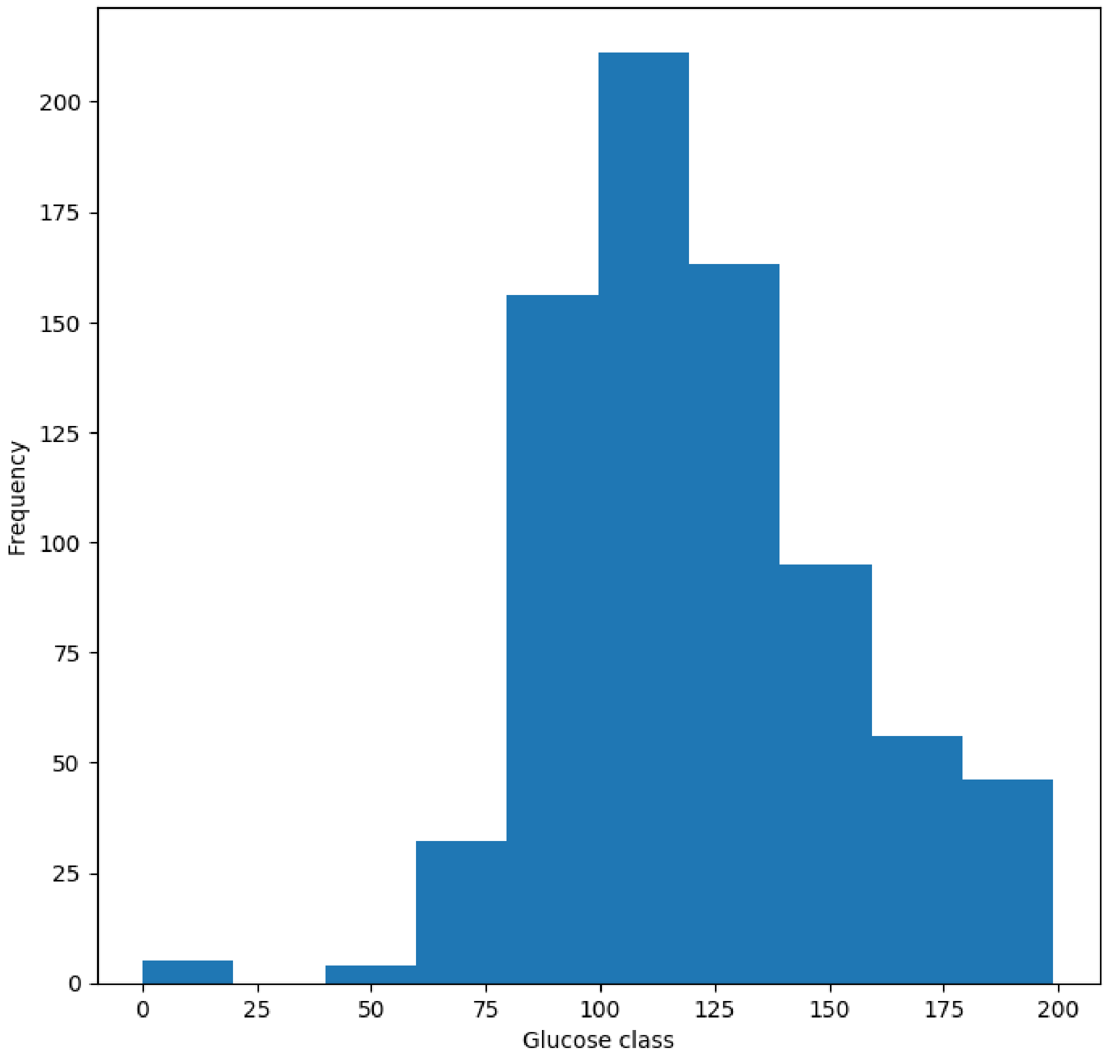
[5]:
```
Pregnancies                  3.367384
Glucose                     31.951796
```

```
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

[6]:
```python
plt.figure(figsize=(8,8),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("Data type of Glucose variable is :",df["Glucose"].dtypes)
```

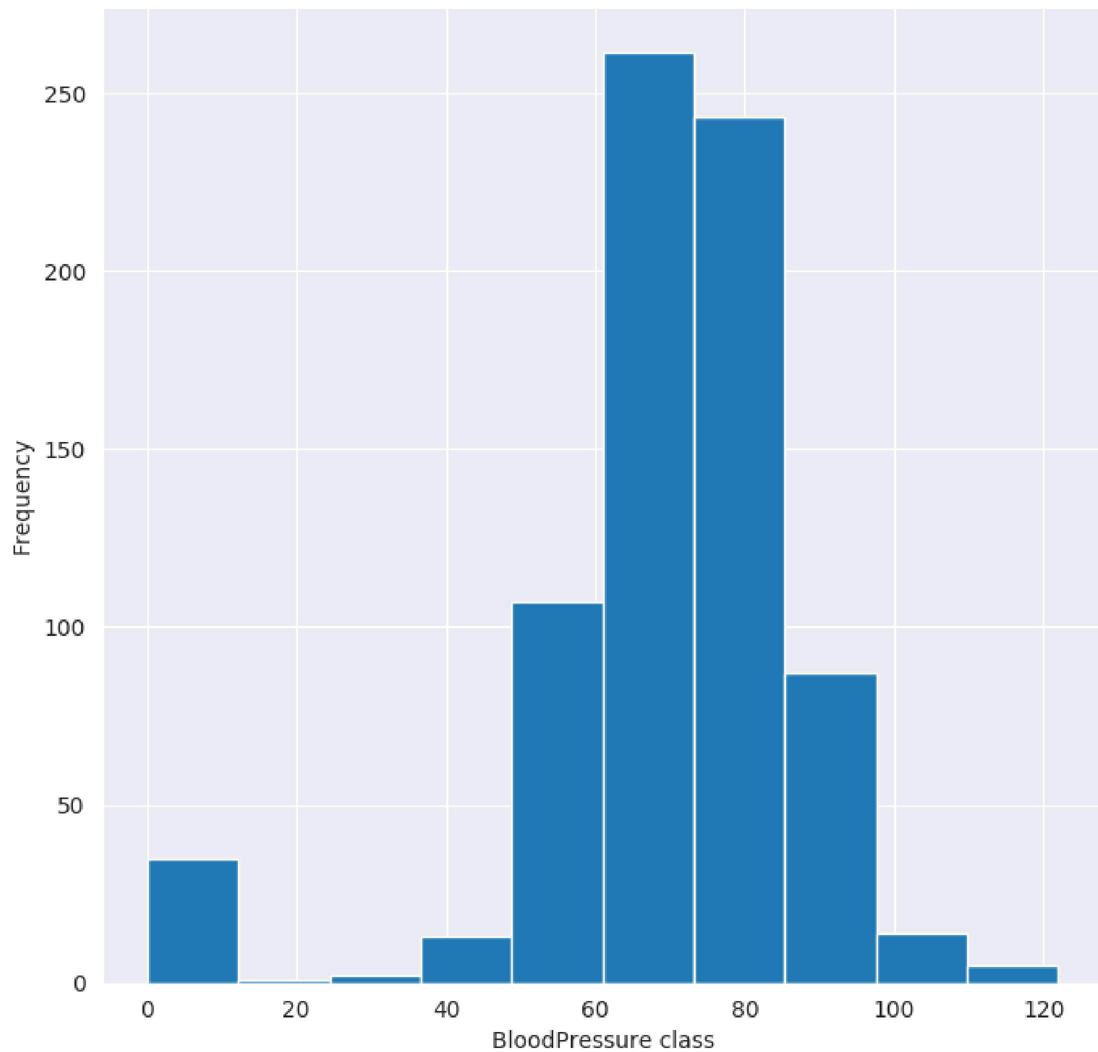
```
Mean of Glucose level is :- 120.89453125
Data type of Glucose variable is : int64
```



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

[8]:
```
plt.figure(figsize=(8,8),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("Data type of BloodPressure variable is :",df["BloodPressure"].dtypes)
```

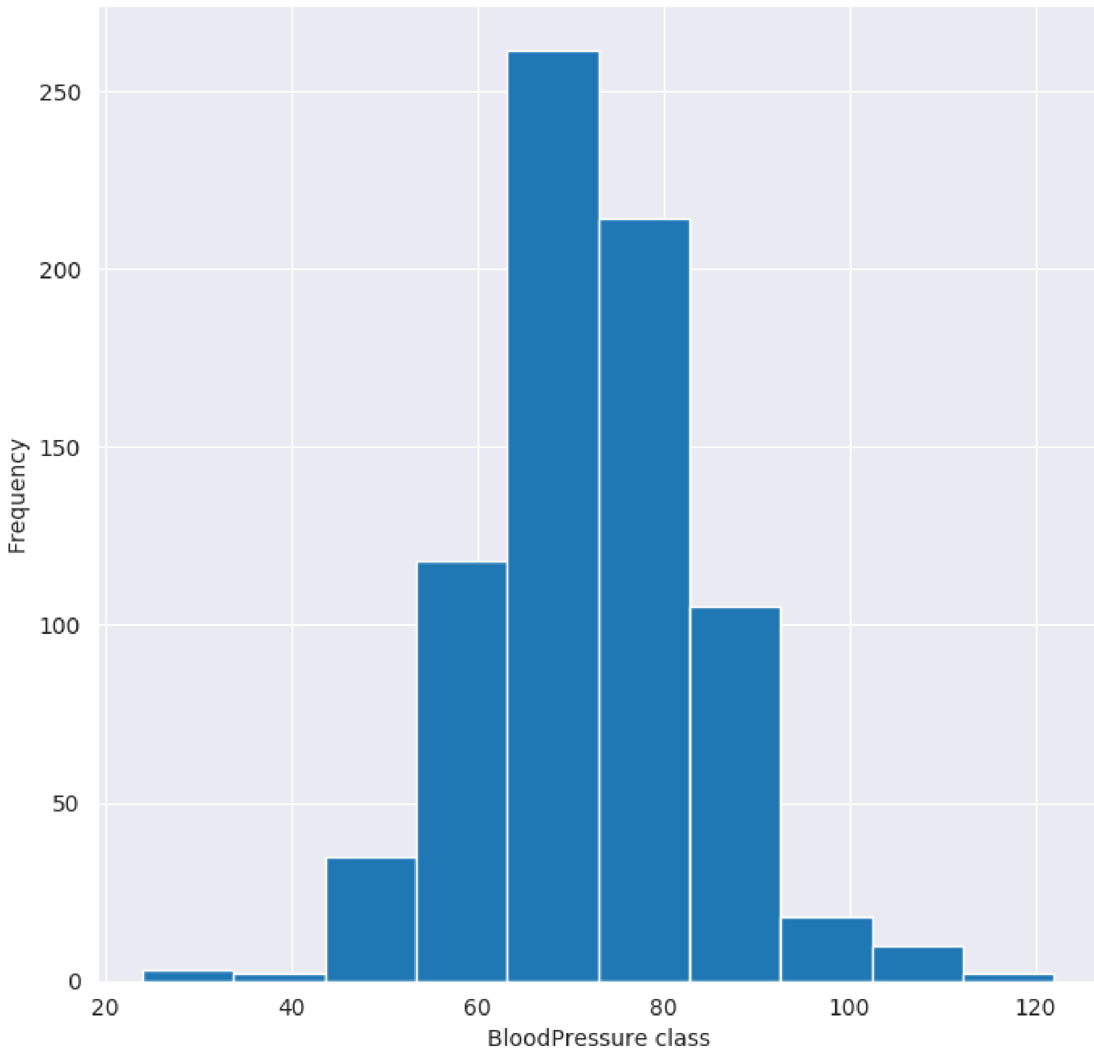
```
Mean of BloodPressure level is :- 69.10546875
Data type of BloodPressure variable is : int64
```



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

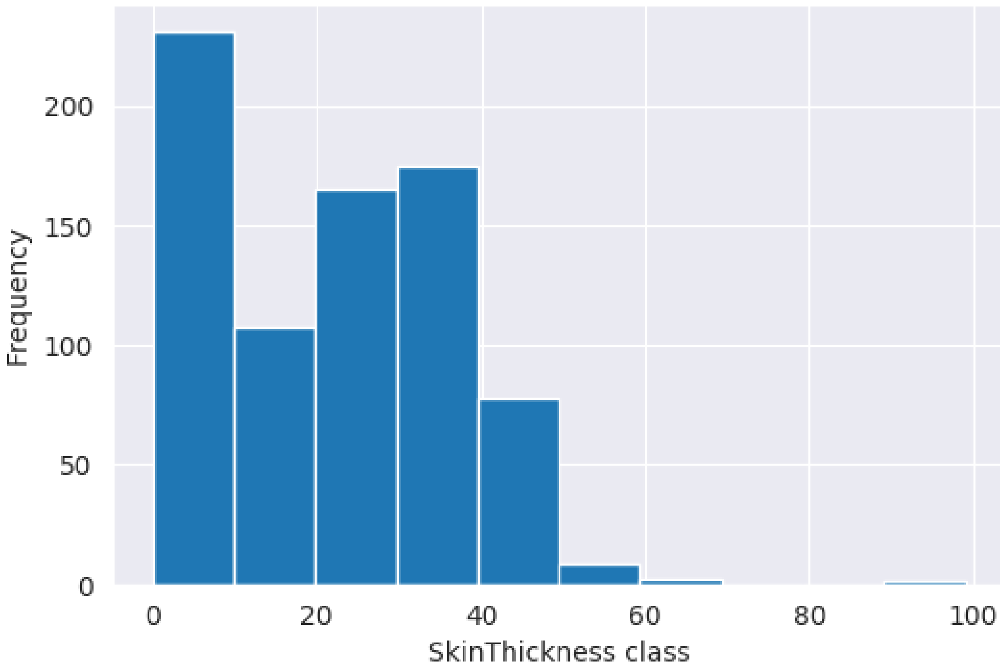
[10]:
```
plt.figure(figsize=(8,8),dpi=100)
plt.xlabel("BloodPressure class")
df["BloodPressure"].plot.hist()
sns.set_style(style="darkgrid")
print("Mean of BloodPressure level is :-",df["BloodPressure"].mean())
print("Data type of BloodPressure variable is :",df["BloodPressure"].dtypes)
```

```
Mean of BloodPressure level is :- 72.25480651855469
Data type of BloodPressure variable is : float64
```



[11]:
```
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel("SkinThickness class")
df["SkinThickness"].plot.hist()
sns.set_style(style="darkgrid")
print("Mean of SkinThickness level is :-",df["SkinThickness"].mean())
print("Data type of SkinThickness variable is :",df["SkinThickness"].dtypes)
```

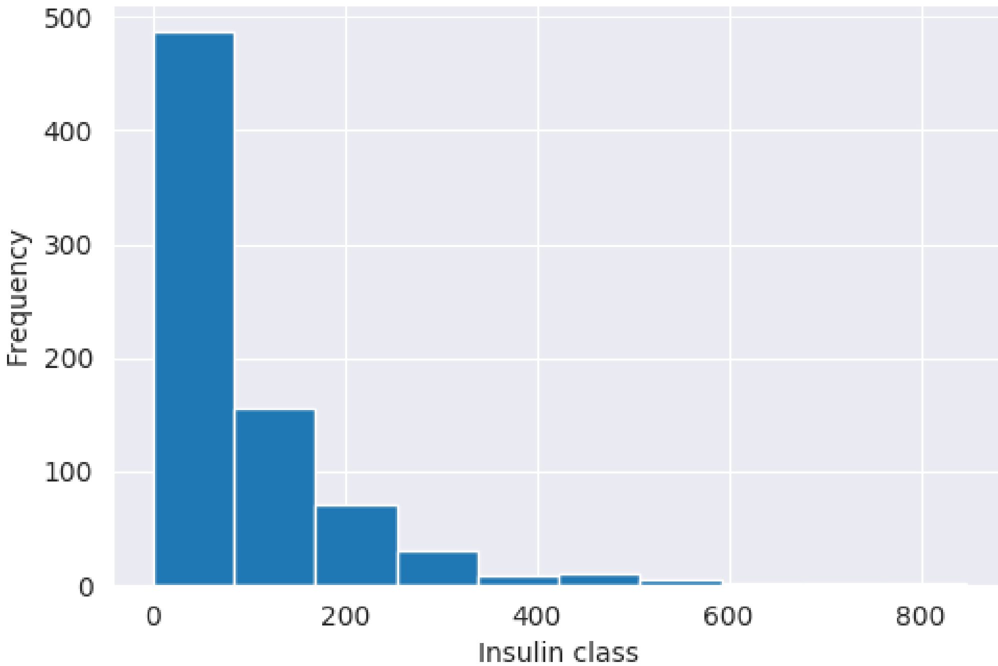
```
Mean of SkinThickness level is :- 20.536458333333332
Data type of SkinThickness variable is : int64
```



[12]:
```
df["SkinThickness"]=df["SkinThickness"].replace(0,df["SkinThickness"].mean())
```

[13]:
```
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel("Insulin class")
df["Insulin"].plot.hist()
sns.set_style(style="darkgrid")
print("Mean of Insulin level is :-",df["Insulin"].mean())
print("Data type of Insulin variable is :",df["Insulin"].dtypes)
```

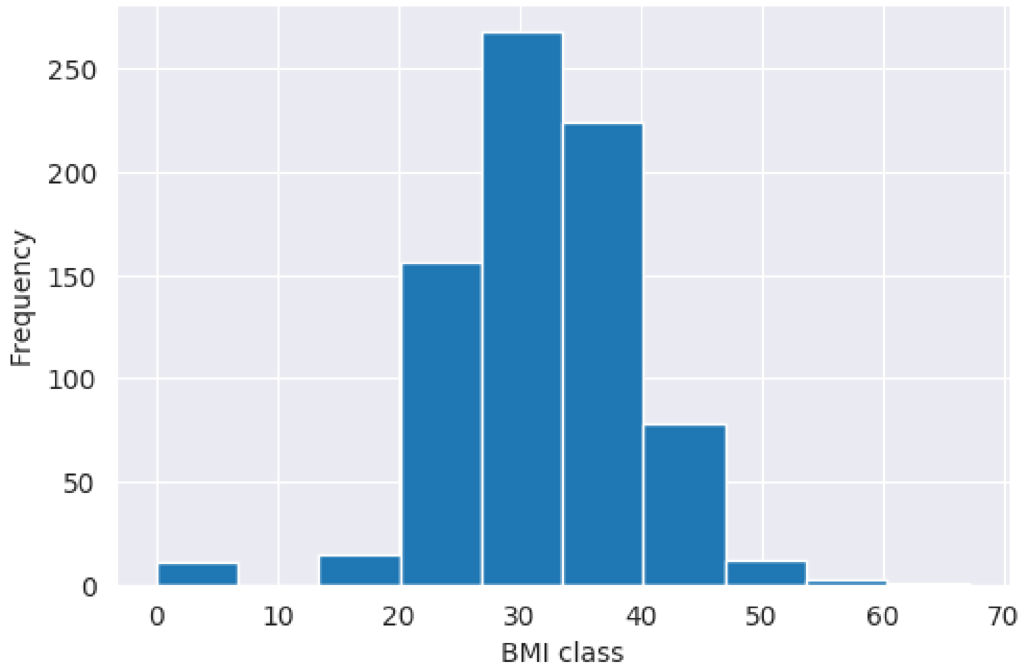
```
Mean of Insulin level is :- 79.79947916666667
Data type of Insulin variable is : int64
```



[14]:
```
df["Insulin"]=df["Insulin"].replace(0,df["Insulin"].mean())
```

[15]:
```
plt.figure(figsize=(6,4),dpi=100)
plt.xlabel("BMI class")
df["BMI"].plot.hist()
sns.set_style(style="darkgrid")
print("Mean of BMI level is :-",df["BMI"].mean())
print("Data type of BMI variable is :",df["BMI"].dtypes)
```

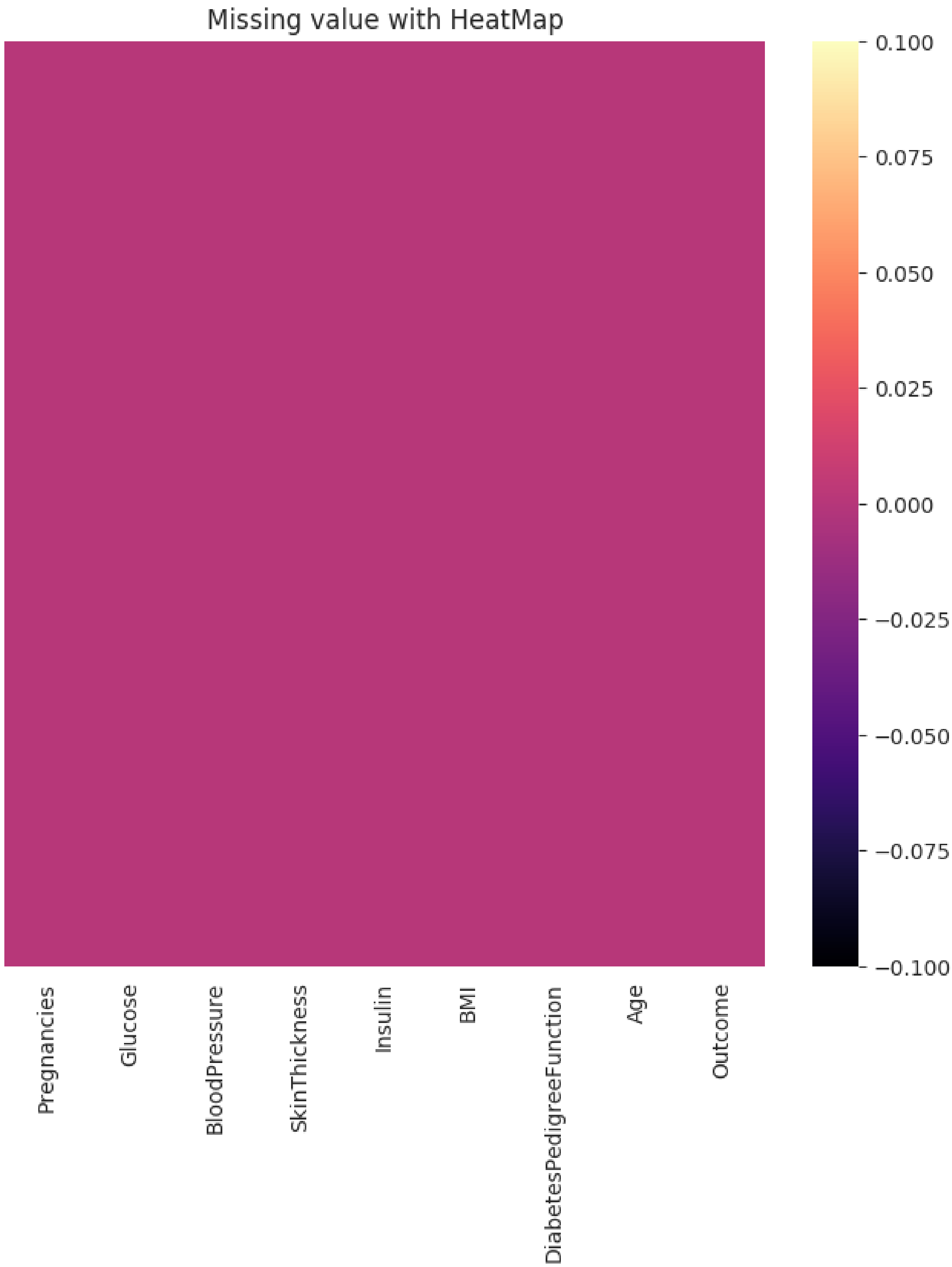
```
Mean of BMI level is :- 31.992578124999977
Data type of BMI variable is : float64
```



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

[17]:
```
plt.figure(figsize=(8,8),dpi=100)
plt.title("Missing value with HeatMap")
sns.heatmap(df.isnull(),cmap="magma",yticklabels=False)
```

[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5bd0febed0>


Missing value with HeatMap
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

[18]:

```
df.head()
```

[18]:

```
   Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI  \
0            6    148.0           72.0      35.000000   79.799479  33.6
1            1     85.0           66.0      29.000000   79.799479  26.6
2            8    183.0           64.0      20.536458   79.799479  23.3
3            1     89.0           66.0      23.000000   94.000000  28.1
4            0    137.0           40.0      35.000000  168.000000  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
```

[19]:

```
df.tail()
```

[19]:

```
     Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI  \
763           10    101.0           76.0      48.000000  180.000000  32.9
764            2    122.0           70.0      27.000000   79.799479  36.8
765            5    121.0           72.0      23.000000  112.000000  26.2
766            1    126.0           60.0      20.536458   79.799479  30.1
767            1     93.0           70.0      31.000000   79.799479  30.4

     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

[20]:

```
df.to_csv("after_week1.csv",index=False)
```

[21]:

```
df.head()
```

[21]:

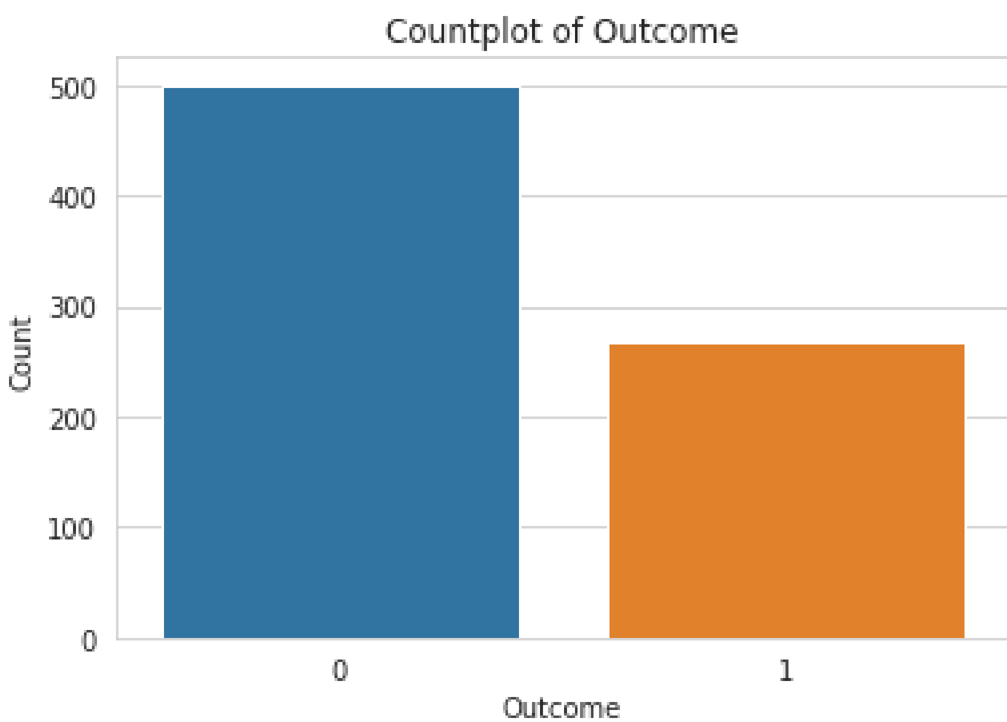
```
   Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI  \
0            6    148.0           72.0      35.000000   79.799479  33.6
1            1     85.0           66.0      29.000000   79.799479  26.6
2            8    183.0           64.0      20.536458   79.799479  23.3
3            1     89.0           66.0      23.000000   94.000000  28.1
4            0    137.0           40.0      35.000000  168.000000  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
```

1.2 Week 2 Task (Health care): Data Exploration:

CountPlot

[22]:
```
sns.set_style("whitegrid")
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\:n 0    500
1    268
Name: Outcome, dtype: int64
```



2 Scatter Plot

[23]:
```
sns.pairplot(df)
plt.title("Scatter Plot between Variable")
```

[23]: Text(0.5, 1, 'Scatter Plot between Variable')

[24]:
```
df.corr()
```

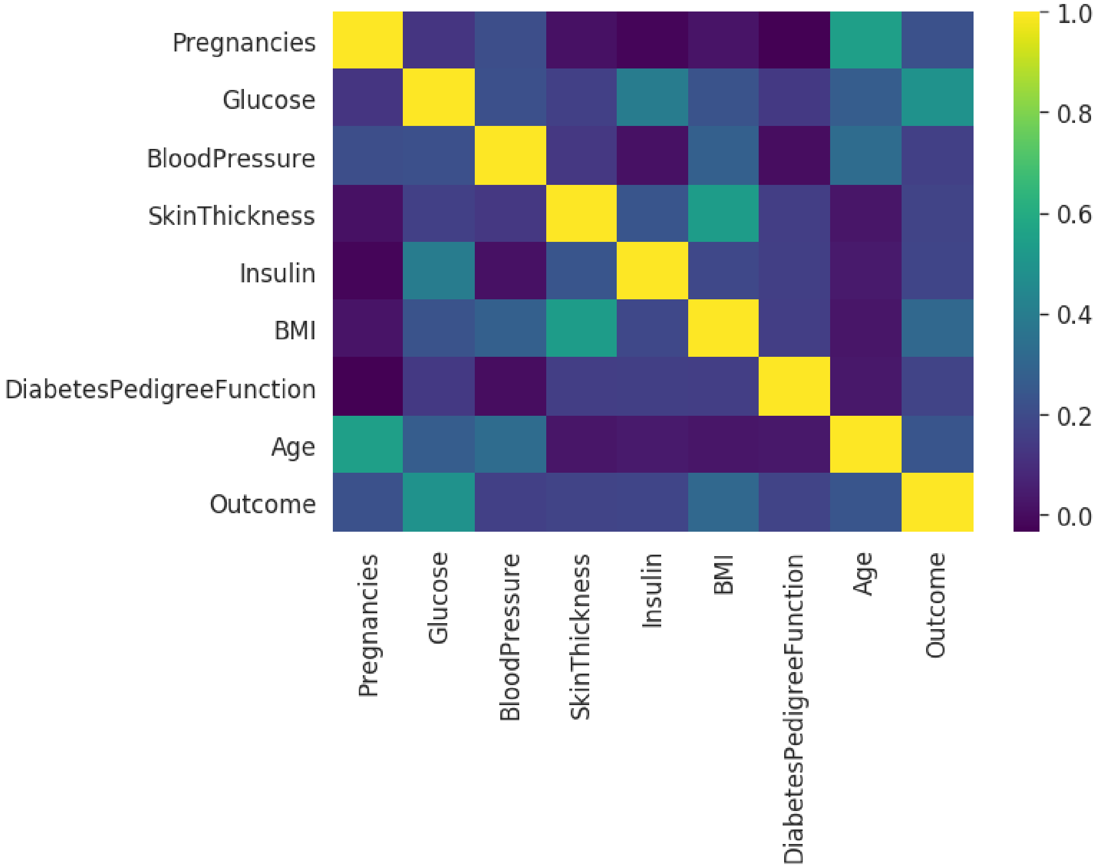
[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	\
Pregnancies	1.000000	0.127964	0.208984	0.013376	
Glucose	0.127964	1.000000	0.219666	0.160766	
BloodPressure	0.208984	0.219666	1.000000	0.134155	
SkinThickness	0.013376	0.160766	0.134155	1.000000	
Insulin	-0.018082	0.396597	0.010926	0.240361	
BMI	0.021546	0.231478	0.281231	0.535703	
DiabetesPedigreeFunction	-0.033523	0.137106	0.000371	0.154961	
Age	0.544341	0.266600	0.326740	0.026423	
Outcome	0.221898	0.492908	0.162986	0.175026	

```
                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.018082  0.021546                 -0.033523
Glucose                   0.396597  0.231478                  0.137106
BloodPressure             0.010926  0.281231                  0.000371
SkinThickness             0.240361  0.535703                  0.154961
Insulin                   1.000000  0.189856                  0.157806
BMI                       0.189856  1.000000                  0.153508
DiabetesPedigreeFunction  0.157806  0.153508                  1.000000
Age                       0.038652  0.025748                  0.033561
Outcome                   0.179185  0.312254                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.266600  0.492908
BloodPressure             0.326740  0.162986
SkinThickness             0.026423  0.175026
Insulin                   0.038652  0.179185
BMI                       0.025748  0.312254
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

[25]:
```
plt.figure(dpi=120)
sns.heatmap(df.corr(),cmap='viridis')
```

[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5bca6386d0>



3 Week 3 Task (Health Care) : Data Modeling:

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

[26]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI  \
0            6    148.0           72.0      35.000000   79.799479  33.6
1            1     85.0           66.0      29.000000   79.799479  26.6
2            8    183.0           64.0      20.536458   79.799479  23.3
3            1     89.0           66.0      23.000000   94.000000  28.1
4            0    137.0           40.0      35.000000  168.000000  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
```

[27]:
```
x=df.iloc[:,:-1].values
y=df.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.
 ↪20,random_state=0)
```

[29]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

[30]:
```
from sklearn.preprocessing import StandardScaler
Scale=StandardScaler()
x_train_std=Scale.fit_transform(x_train)
x_test_std=Scale.transform(x_test)
norm=lambda a:(a-min(a))/(max(a)-min(a))
df_norm=df.iloc[:,:-1]
```

[31]:
```
df_normalized=df_norm.apply(norm)
x_train_norm,x_test_norm,y_train_norm,y_test_norm=train_test_split(df_normalized.
 ↪values,y,test_size=0.20,random_state=0)
```

[32]:
```
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,)
```

[33]:
```
#Data is mostly numerical and in such scenario , Logistic Regression works fine.
```

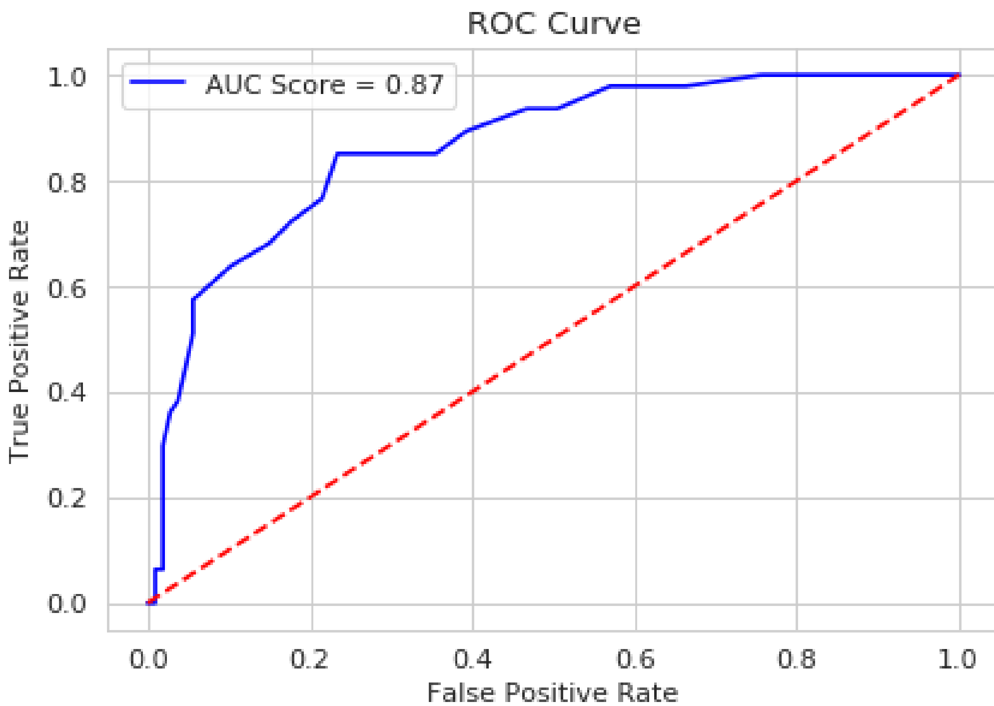
KNN With Standard Scaling

[34]:
```
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
knn_model=KNeighborsClassifier(n_neighbors=25)
knn_model.fit(x_train_std,y_train)
knn_pred=knn_model.predict(x_test_std)
```

[35]:
```python
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
```

[35]:
```
<matplotlib.legend.Legend at 0x7f5bc05e28d0>
```



KNN With Normalization

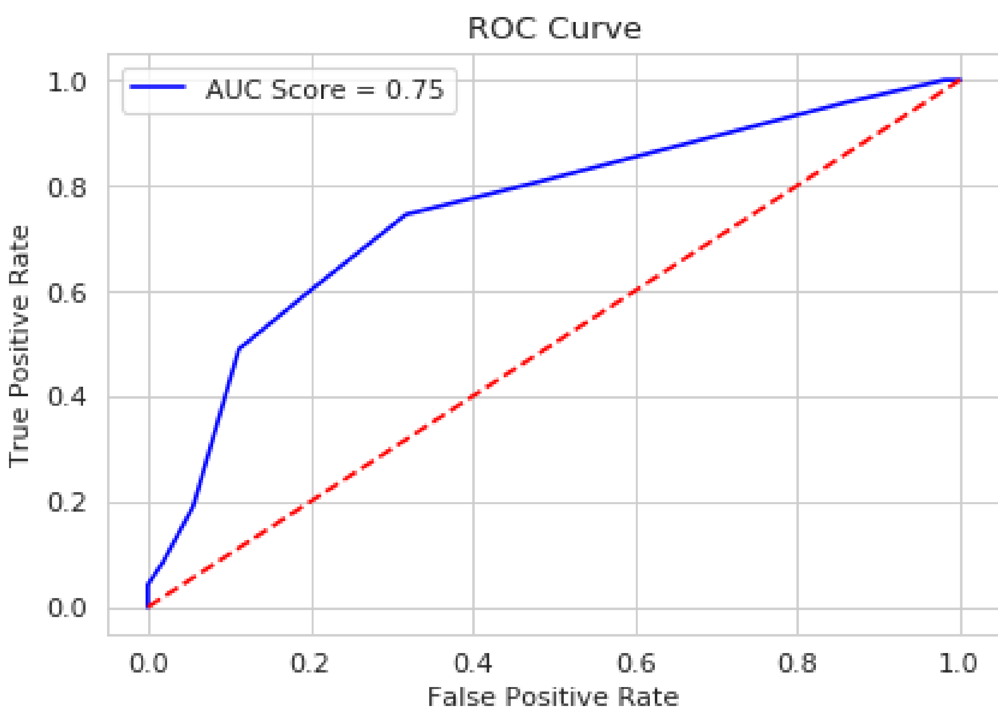
[36]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model_norm = KNeighborsClassifier(n_neighbors=25)
#Using 25 Neighbors just as thumb rule sqrt of observation
knn_model_norm.fit(x_train_norm,y_train_norm)
knn_pred_norm=knn_model_norm.predict(x_test_norm)
```

[37]:
```
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
```

[37]: `<matplotlib.legend.Legend at 0x7f5bc05319d0>`



We can clearly see that KNN with Standardization is better than Normalization

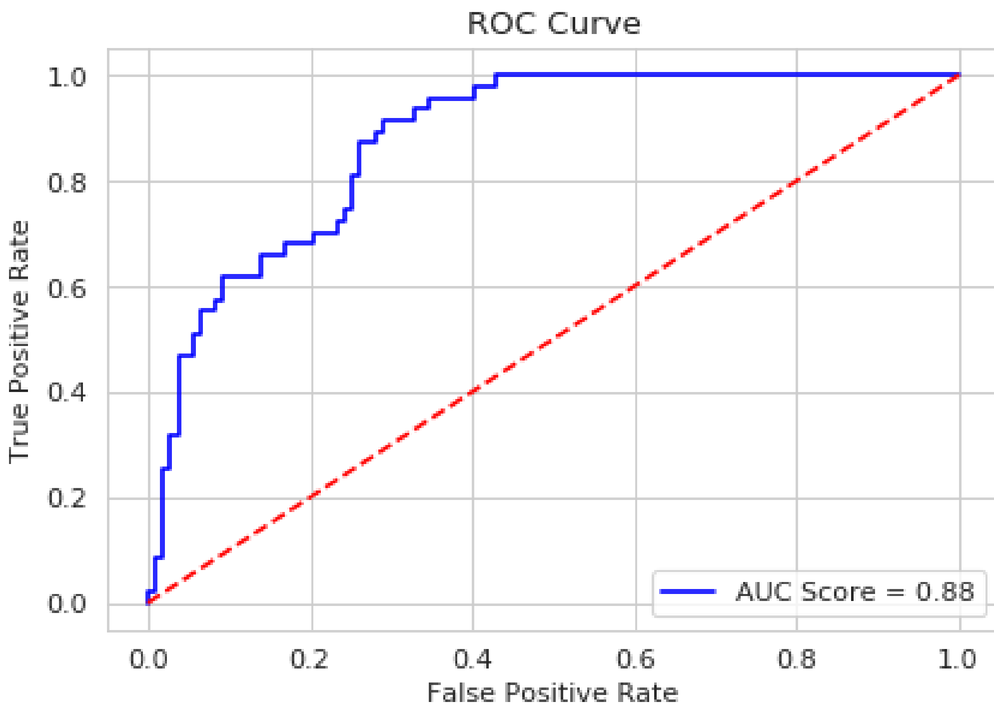
Support Vectore Classifier

[38]:
```
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)
```

[39]:
```
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
```

[39]:
```
<matplotlib.legend.Legend at 0x7f5bc05a3310>
```



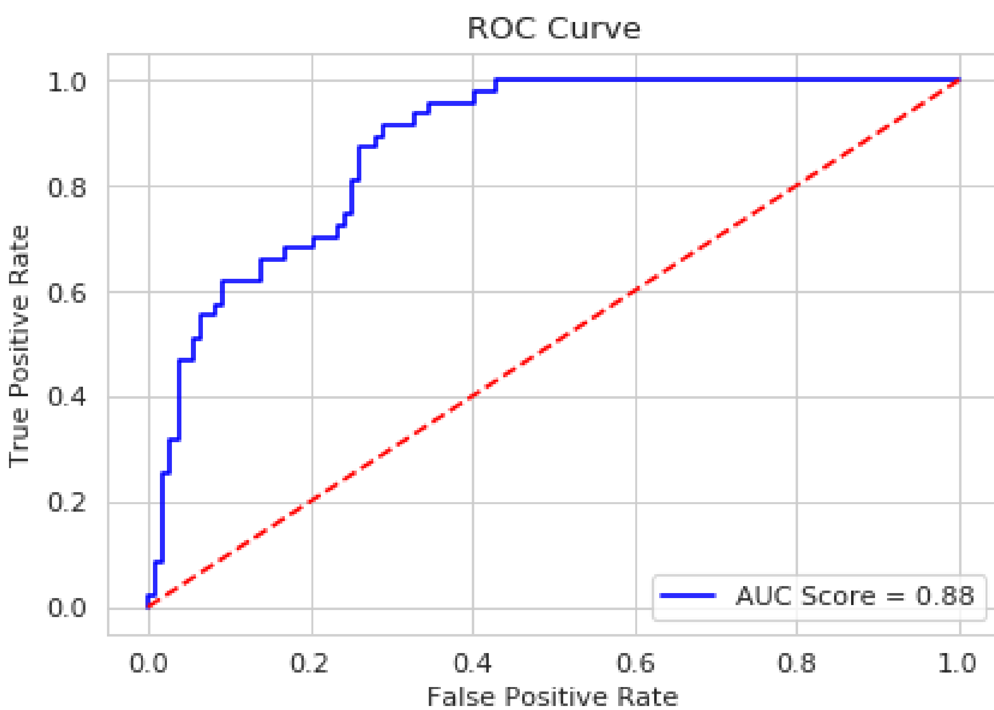
[40]:
```
from sklearn.svm import SVC
svc_model_rbf = SVC(kernel='rbf',random_state=0,probability=True,C=1)
svc_model_rbf.fit(x_train_std,y_train)
svc_pred_rbf=svc_model_rbf.predict(x_test_std)
```

[41]:
```
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
```

[41]: <matplotlib.legend.Legend at 0x7f5bc0531410>



SVC with Linear Kernel is better than RBF Kernel,

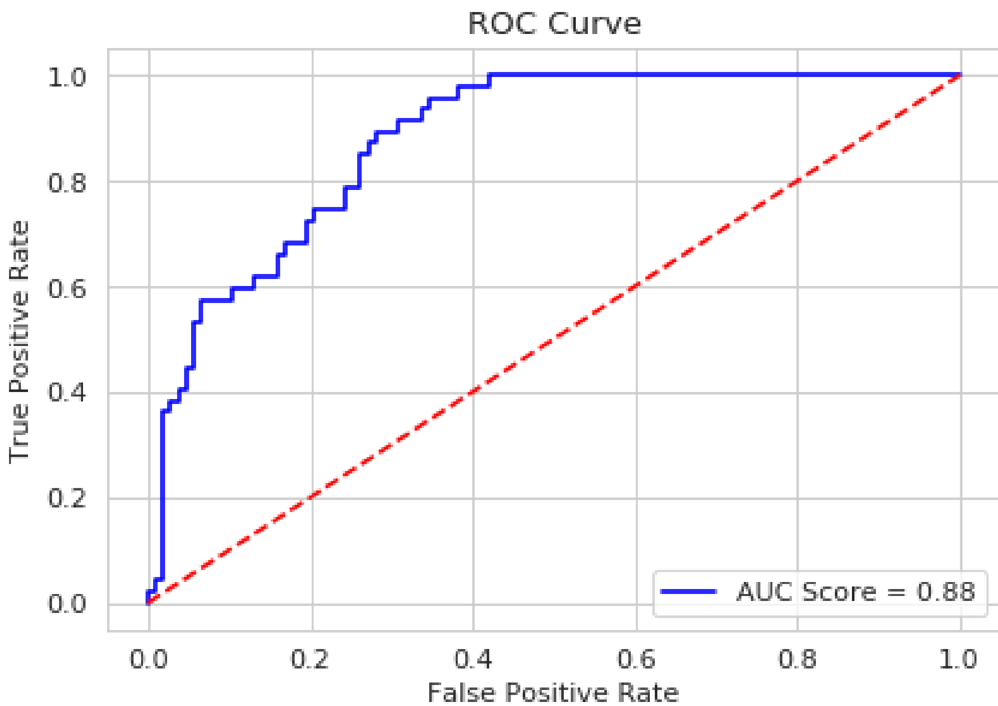
Logistic Regression

[42]:
```python
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)
```

[43]:
```python
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
```

[43]: <matplotlib.legend.Legend at 0x7f5bc04220d0>



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

Ensemble Learning(RF)

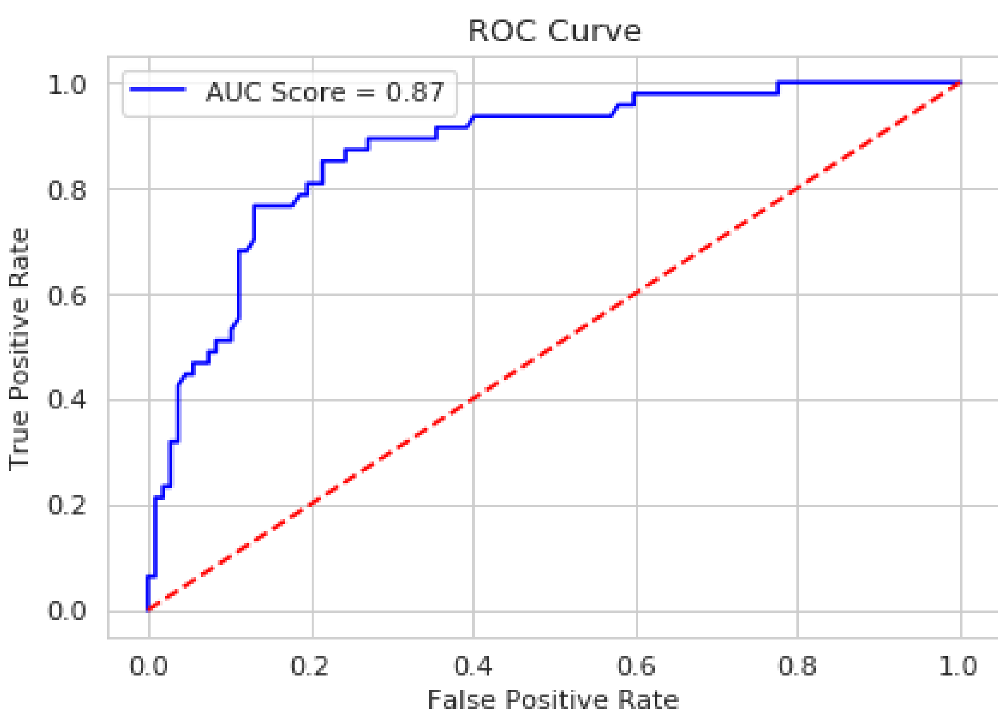
[44]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=1000,random_state=0)
rf_model.fit(x_train_std,y_train)
rf_pred=rf_model.predict(x_test_std)
```

[45]:
```
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
```

[45]: <matplotlib.legend.Legend at 0x7f5bc0391c50>



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

4 Project Task: Week 4

Data Reporting:

a. Pie chart to describe the diabetic or non-diabetic population

b. Scatter charts between relevant variables to analyze the relationships

c. Histogram or frequency charts to analyze the distribution of the data

d. Heatmap of correlation analysis among the relevant variables

e. Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.

Please Find the Below Link to view or Dowload the Tabluae DashBoard Report.

5 https://prod-apnortheast-a.online.tableau.com/#/site/darshanan/workbo

```
[ ]:
```

6 Thank you..!

```
[ ]:
```