Capstone Project 2 Healthcare

DESCRIPTION

Problem Statement NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases. The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Dataset Description The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Variables Description Pregnancies Number of times pregnant

- Glucose Plasma glucose concentration in an oral glucose tolerance test
- BloodPressure Diastolic blood pressure (mm Hg)
- SkinThickness Triceps skinfold thickness (mm)
- Insulin Two hour serum insulin
- BMI Body Mass Index
- DiabetesPedigreeFunction Diabetes pedigree function
- Age Age in years
- Outcome Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

Project Task: Week 1

Data Exploration:

1.Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value: • Glucose • BloodPressure • SkinThickness • Insulin • BMI

2.Visually explore these variables using histograms. Treat the missing values accordingly.

3.There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
df=pd.read_csv('health care diabetes.csv')
df.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	C
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 [4]:
```
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]:
```
df.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
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	
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	

Insight from Descriptive Analysis

- There are 768 observations of 9 variables.
- Independent variables are Pregnencies , Glucose, BloodPressure, Insulin, BMI and DiabetesPedigree Function.
- Age is Outcome Variable.
- Average Age of Patients is 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 and DPF = 0.471876
- Variation (Standard Deviation) in variables can be easily observed from table below :->

In [72]:
```
print("Standard Deviation of Each Variable ==> ")
df.apply(np.std)
```

```
Standard Deviation of Each Variable ==>
```

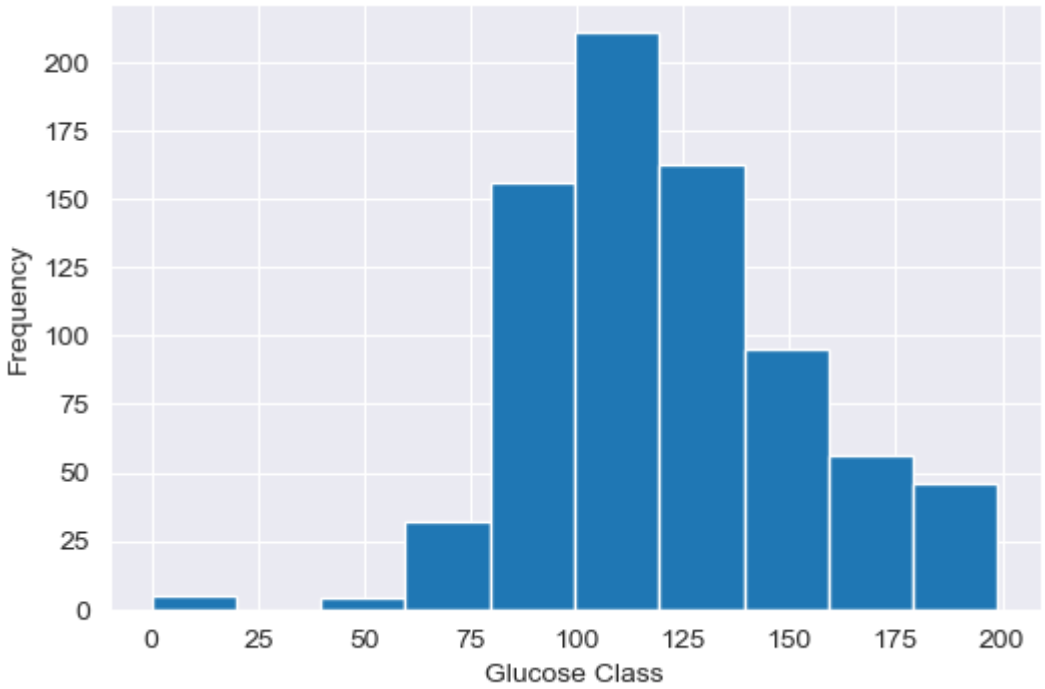
Out[72]:
```
Pregnancies                 3.367384
Glucose                    31.951796
BloodPressure              19.343202
SkinThickness              15.941829
Insulin                   115.168949
BMI                         7.879026
DiabetesPedigreeFunction    0.331113
Age                        11.752573
Outcome                     0.476641
dtype: float64
```

Treating Missing Values and Analysing Distribution of Data

Note that In question no.3 of week 1, We have to plot frequency of given variable that is similar to histogram.

In [73]:
```
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
```

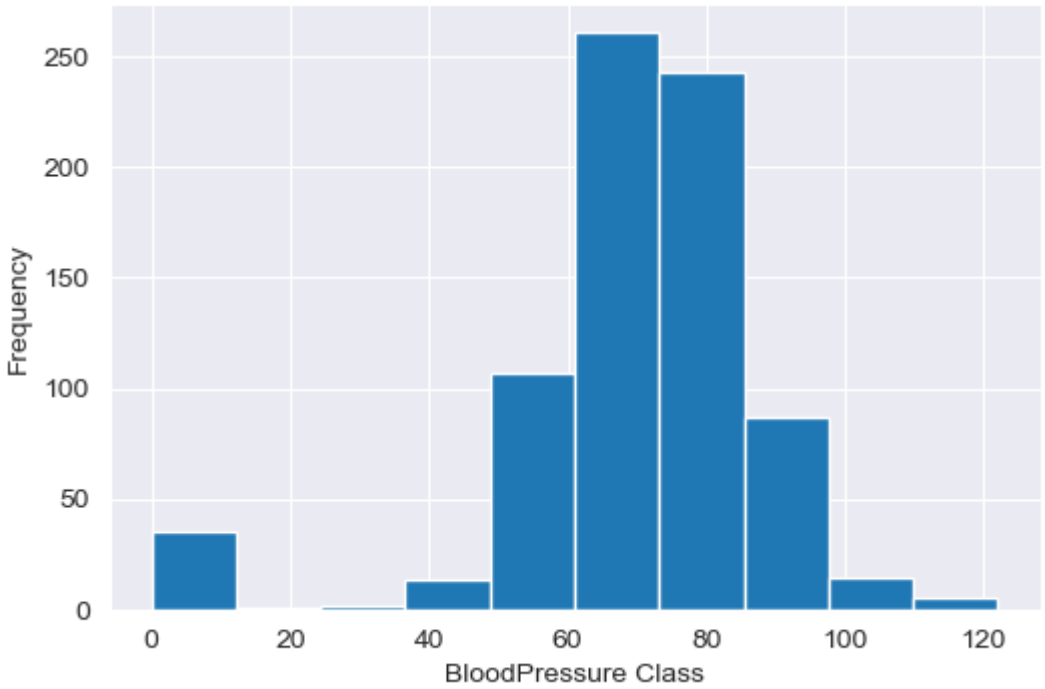


Treating the missing values which are basically 0 by mean of Glucose level. This is because from histogram most of observation have Glucose level between 100 and 120.

In [94]:
```
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
```

In [95]:
```
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
```

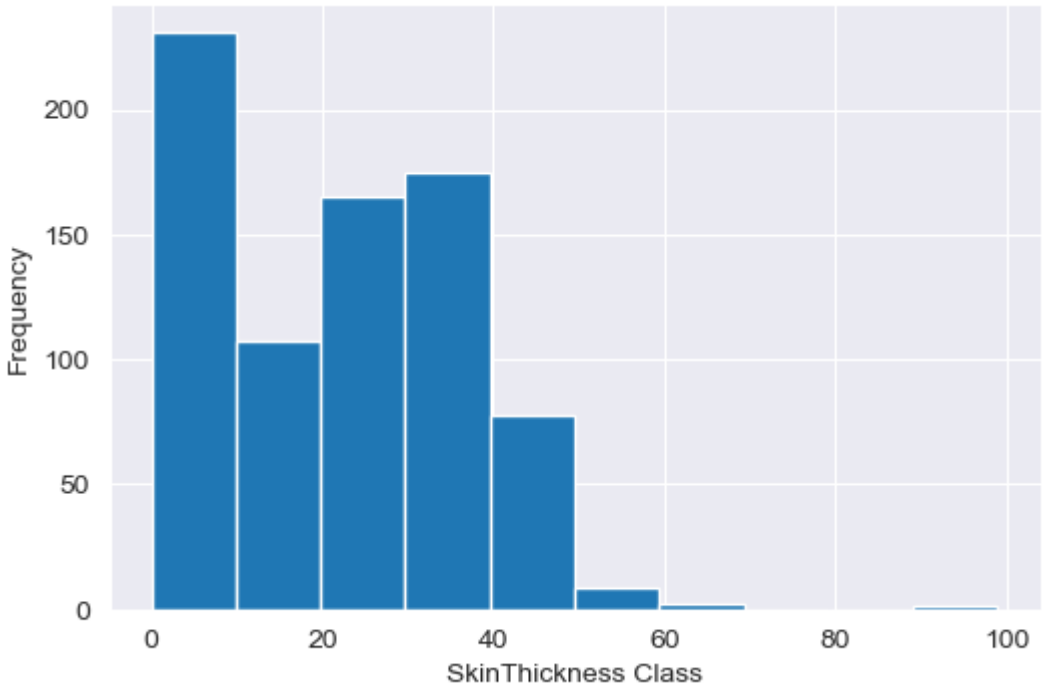


Treating missing values which are basically 0 by mean of BloodPressure level. This is because we can see from histogram most of observation have BP level between 70 and 80.

In [96]:
```
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())
```

In [97]:
```
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
```

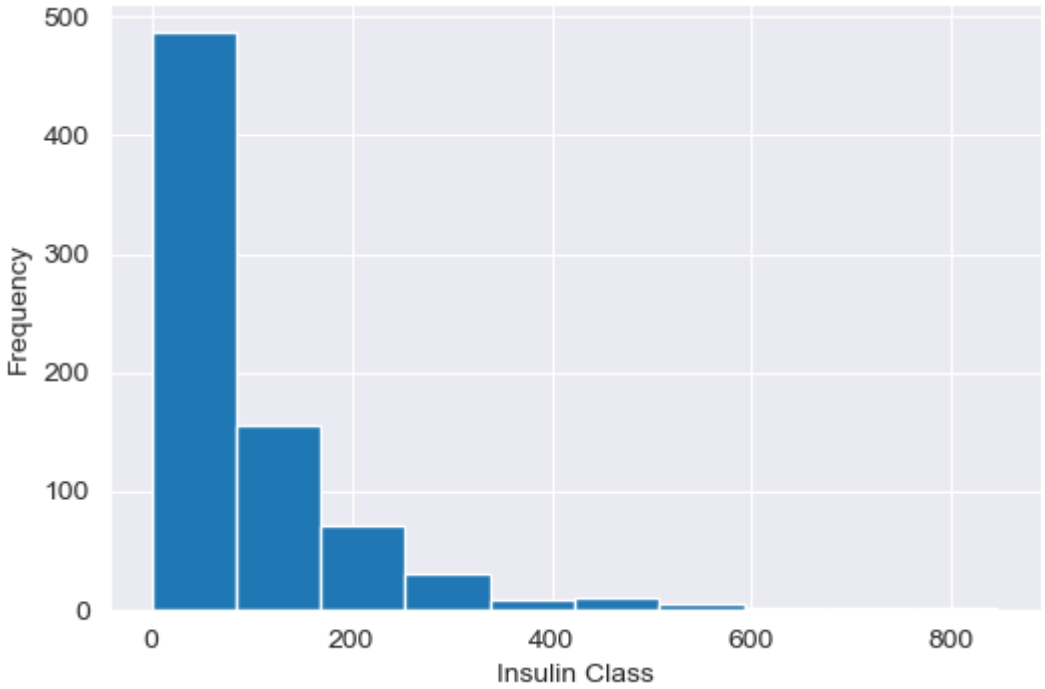


Treating missing values which are basically 0 by mean of Skin Thickness. This is because we can see from histogram most of observation have Skin Thickness between 20 and 30.

In [98]:
```
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].mean())
```

In [99]:
```
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)
```

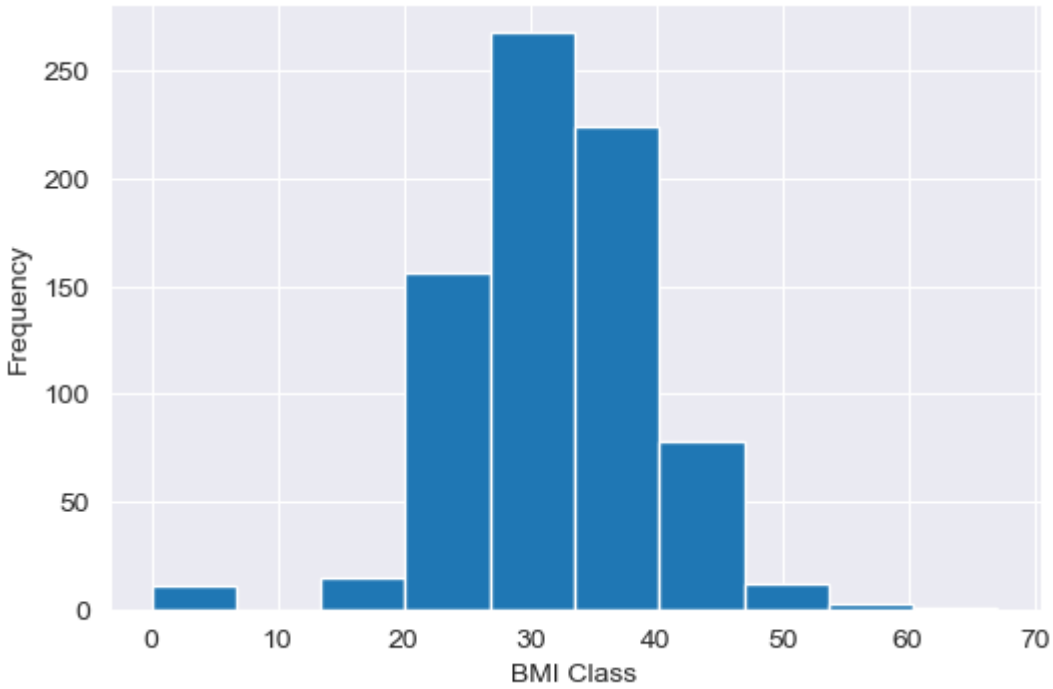
```
Mean of Insulin is :- 79.79947916666667
Datatype of Insulin Variable is: int64
```



In [100]:
```
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].mean())
```

In [101]:
```
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)
```

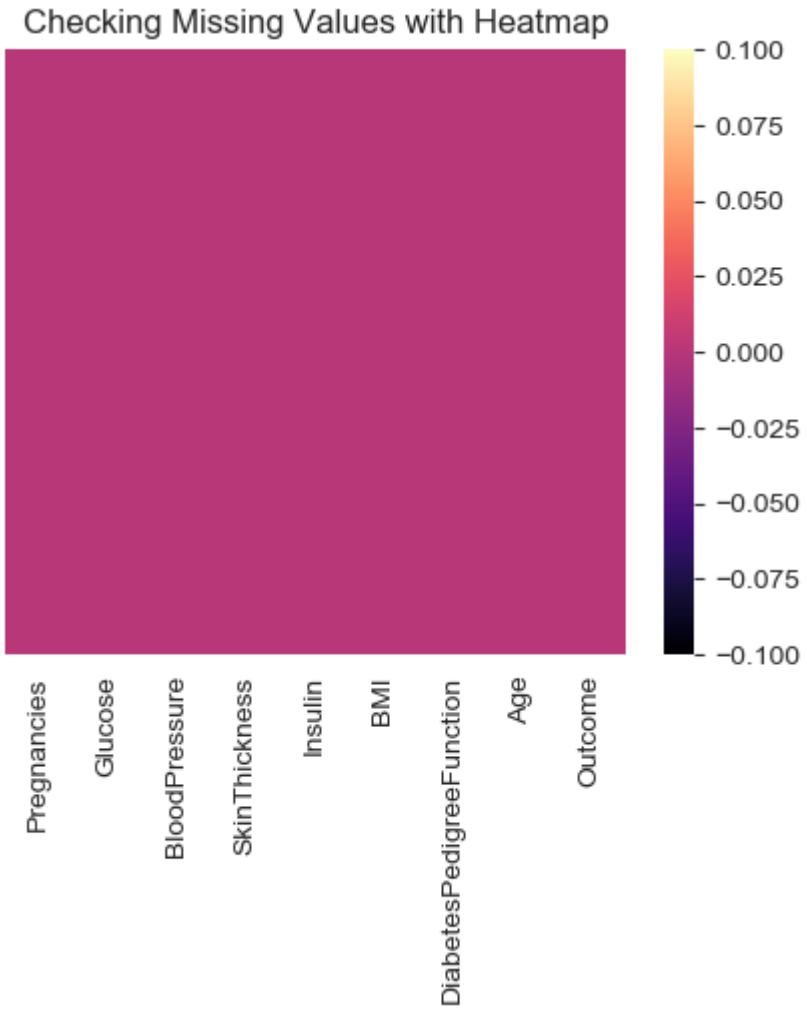
```
Mean of BMI is :- 31.992578124999977
Datatype of BMI Variable is: float64
```



In [102]:
```
df['BMI']=df['BMI'].replace(0,df['BMI'].mean())
```

In [103]:
```
plt.figure(figsize=(5,4),dpi=100)
plt.title('Checking Missing Values with Heatmap')
sns.heatmap(df.isnull(),cmap='magma',yticklabels=False)
```

Out[103]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21de8350448>
```



In [104]:
```
df.head()
```

Out[104]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	5
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	3
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	3
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	2
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	3

In [105]:
```
df.tail()
```

Out[105]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
763	10	101.0	76.0	48.000000	180.000000	32.9	0.171	
764	2	122.0	70.0	27.000000	79.799479	36.8	0.340	
765	5	121.0	72.0	23.000000	112.000000	26.2	0.245	
766	1	126.0	60.0	20.536458	79.799479	30.1	0.349	
767	1	93.0	70.0	31.000000	79.799479	30.4	0.315	

In [106]:
```
df.to_csv('new_health_care.csv',index=False)
```

In [107]:
```
df.head()
```

Out[107]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	5
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	3
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	3
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	2
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	3

Project Task: Week 2

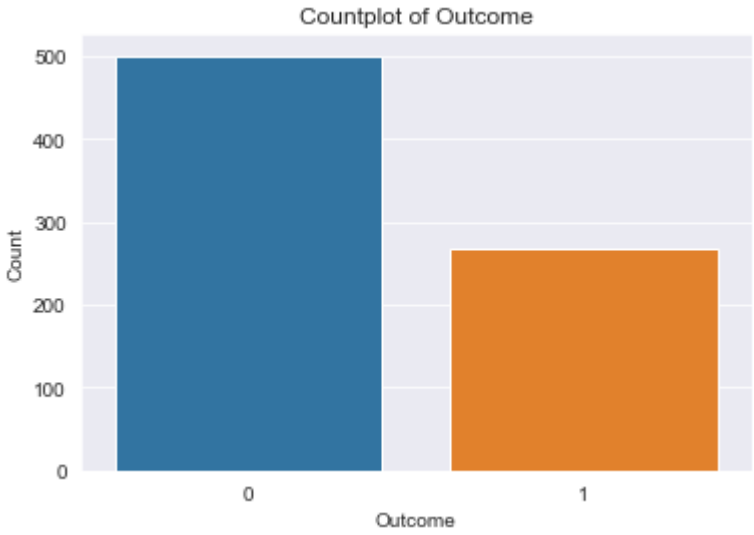
Data Exploration:

1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.
2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
3. Perform correlation analysis. Visually explore it using a heat map.

Countplot

In [108]:
```
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
```

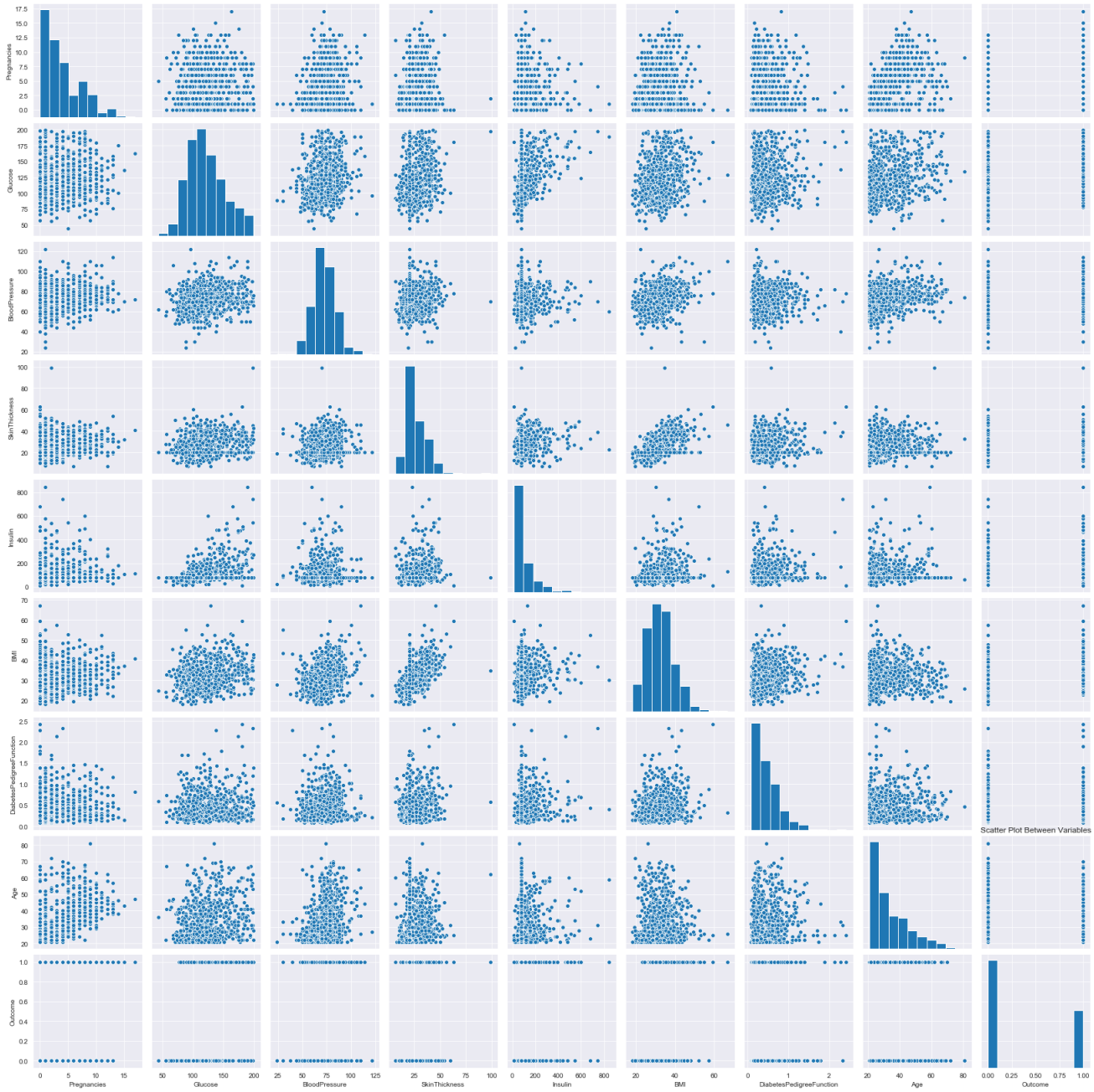


We can see that both class are balanced so we need not perform any sampling method to maintain the balance between both the classes. Therefore we will be directly use this data for training and testing purpose without performing any sampling method. Meanwhile during Model Validation, we also need not worry about ROC Curve because data is not imbalanced, but as this is medical data so we will be using ROC curve to make sure that TYPE 2 ERROR will not be there.

Scatter Plot

In [109]:
```
sns.pairplot(df)
plt.title('Scatter Plot Between Variables')
```

Out[109]:
```
Text(0.5, 1, 'Scatter Plot Between Variables')
```



We can see from the 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. We will explore more when analyzing correlation

Correlation Analysis

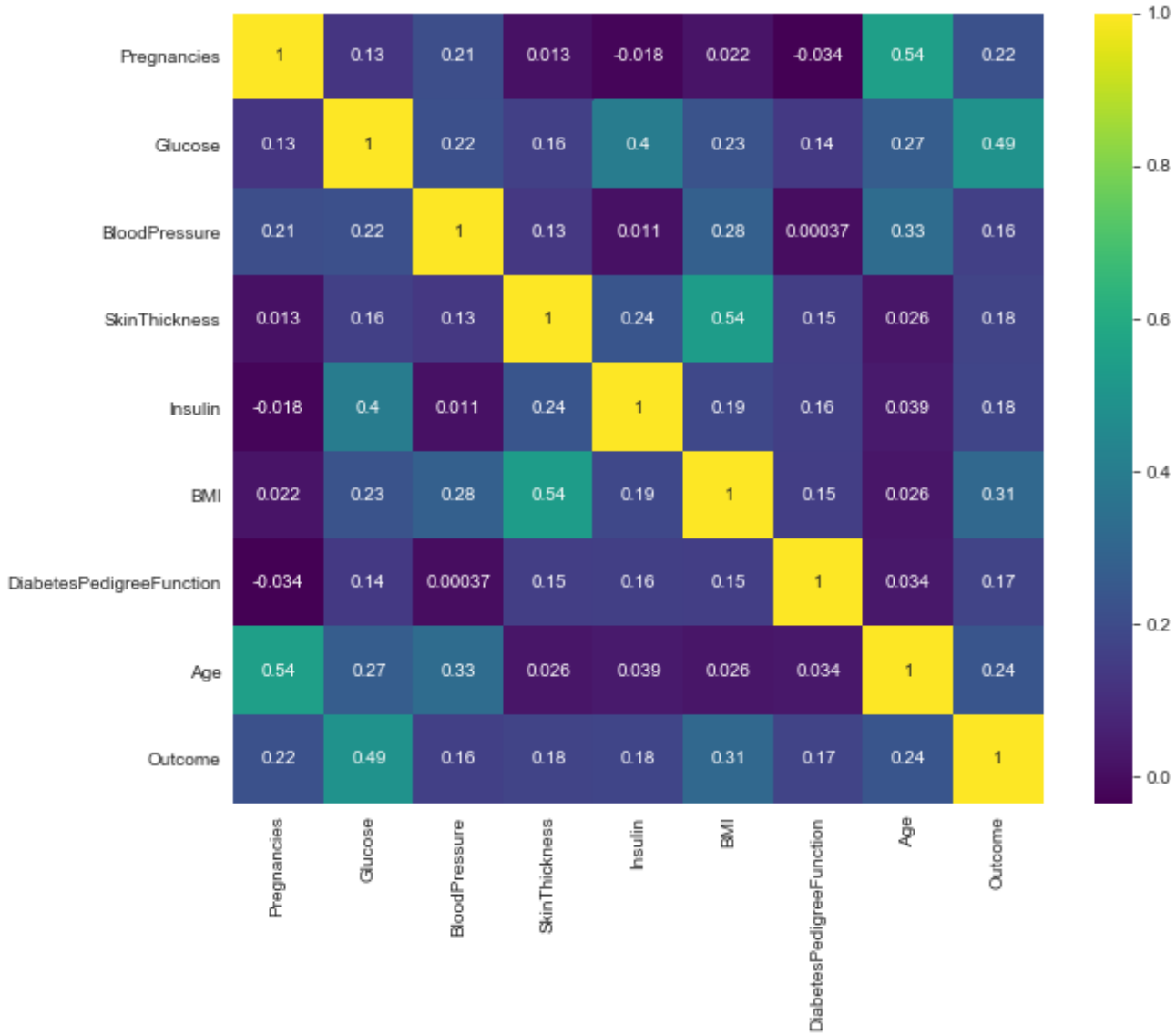
In [110]:
```
df.corr()
```

Out[110]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
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	
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	

In [111]:
```
plt.subplots(figsize=(10,8))
sns.heatmap(df.corr(),annot=True,cmap='viridis')
```

Out[111]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21dea2ce708>
```



Project Task: Week 3

Data Modeling:

1.Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

2.Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

3.Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

Data Preprocessing

In [112]:
```
x=df.iloc[:,:-1].values
y=df.iloc[:,-1].values
```

In [113]:
```
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 [114]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

In [115]:
```
from sklearn.preprocessing import StandardScaler
```

In [116]:
```
Scale=StandardScaler()
x_train_std=Scale.fit_transform(x_train)
x_test_std=Scale.transform(x_test)
```

In [117]:
```
norm=lambda a:(a-min(a))/(max(a)-min(a))
```

In [118]:
```
df_norm=df.iloc[:,:-1]
```

In [119]:
```
df_normalized=df_norm.apply(norm)
```

In [120]:
```
x_train_norm,x_test_norm,y_train_norm,y_test_norm=train_test_split(df_normalized.valu
es,y,test_size=0.20,random_state=0)
```

In [121]:
```
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 here is mostly numerical and in such scenario, Logistic Regression would work fine. We have also seen earlier (in week 2) that variables are depending somewhat linearly on target, hence this is good for Logistic Regression. We will be also be using Support Vector Classifier, Perceptron Learning, Random Forest (Ensemble Learning) to see if we can improve accuracy.

Note that: These learning algorithm also work on linear data very well. To validate the model we will be using train test split, for accuracy we will be using accuracy with confusion matrix because classes are balanced. We 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 here.

KNN

1. KNN With Standard Scaling

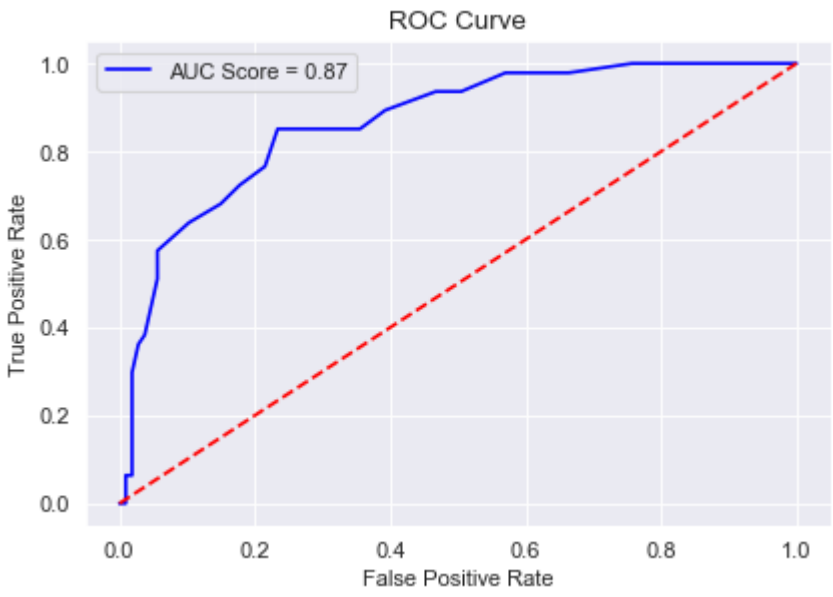
In [122]:
```
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=25)
#Using 25 Neighbors just as thumb rule sqrt of observation
knn_model.fit(x_train_std,y_train)
knn_pred=knn_model.predict(x_test_std)
```

In [123]:

```
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')

confusion = metrics.confusion_matrix(y_test,knn_pred)
print("Confusion Matrix :""\n", confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

print ("\n", "The confusion matrix explained:")
print("\n","True Positives (TP): we correctly predicted that they do have diabetes: "
,TP)
print("\n","True Negatives (TN): we correctly predicted that they don't have diabete
s:", TN)
print("\n","False Positives (FP): we incorrectly predicted that they do have diabetes
(Type I error)",FP)
print("\n","False Negatives (FN): we incorrectly predicted that they don't have diabe
tes (Type II error)",FN)

# Sensitivity
sensitivity = TP / float(FN + TP)
print("\n","Sensitivity :",sensitivity)

#Specificity
specificity = TN / (TN + FP)
print("\n","Specificity:", specificity)

#  AUC (ROC)
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


Confusion Matrix :
 [[96 11]
 [17 30]]

 The confusion matrix explained:

 True Positives (TP): we correctly predicted that they do have diabetes:  30

 True Negatives (TN): we correctly predicted that they don't have diabetes: 96

 False Positives (FP): we incorrectly predicted that they do have diabetes (Type I e
rror) 11

 False Negatives (FN): we incorrectly predicted that they don't have diabetes (Type
II error) 17

 Sensitivity : 0.6382978723404256

 Specificity: 0.897196261682243

 ROC Curve
```

Out[123]: <matplotlib.legend.Legend at 0x21dec450b08>



2. KNN With Normalization

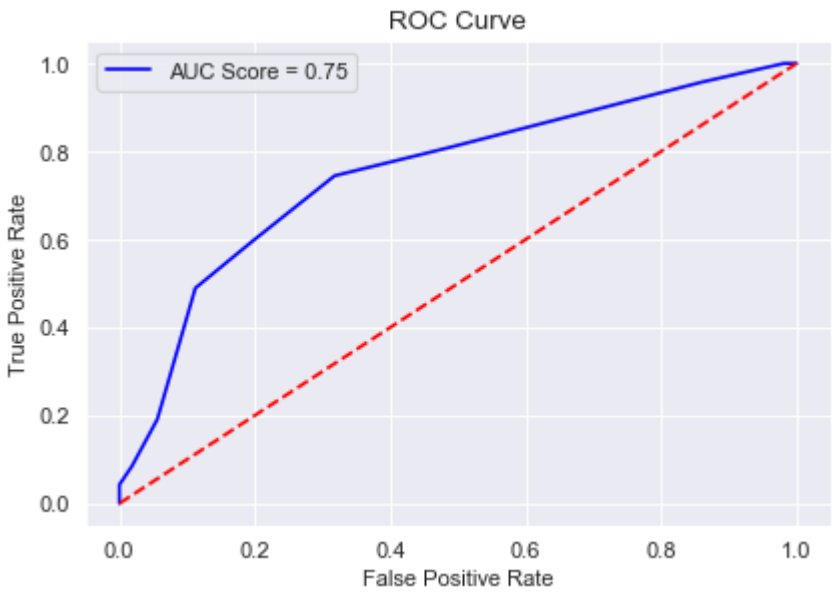
In [124]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model_norm = KNeighborsClassifier(n_neighbors=25)
#Using 25 Neighbors just as thumb rule sqrt of observation
knn_model_norm.fit(x_train_norm,y_train_norm)
knn_pred_norm=knn_model_norm.predict(x_test_norm)
```

In [125]:

```
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')

confusion = metrics.confusion_matrix(y_test,knn_pred_norm)
print("Confusion Matrix :""\n", confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

print ("\n", "The confusion matrix explained:")
print("\n","True Positives (TP): we correctly predicted that they do have diabetes: "
,TP)
print("\n","True Negatives (TN): we correctly predicted that they don't have diabete
s:", TN)
print("\n","False Positives (FP): we incorrectly predicted that they do have diabetes
(Type I error)",FP)
print("\n","False Negatives (FN): we incorrectly predicted that they don't have diabe
tes (Type II error)",FN)

# Sensitivity
sensitivity = TP / float(FN + TP)
print("\n","Sensitivity :",sensitivity)

#Specificity
specificity = TN / (TN + FP)
print("\n","Specificity:", specificity)

#  AUC (ROC)
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


Confusion Matrix :
 [[96 11]
 [15 32]]

 The confusion matrix explained:

 True Positives (TP): we correctly predicted that they do have diabetes:  32

 True Negatives (TN): we correctly predicted that they don't have diabetes: 96

 False Positives (FP): we incorrectly predicted that they do have diabetes (Type I e
rror) 11

 False Negatives (FN): we incorrectly predicted that they don't have diabetes (Type
II error) 15

 Sensitivity : 0.6808510638297872

 Specificity: 0.897196261682243

 ROC Curve
```

Out[125]: <matplotlib.legend.Legend at 0x21dec4bef08>



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

1.Support Vector Classifier with Linear Kernel

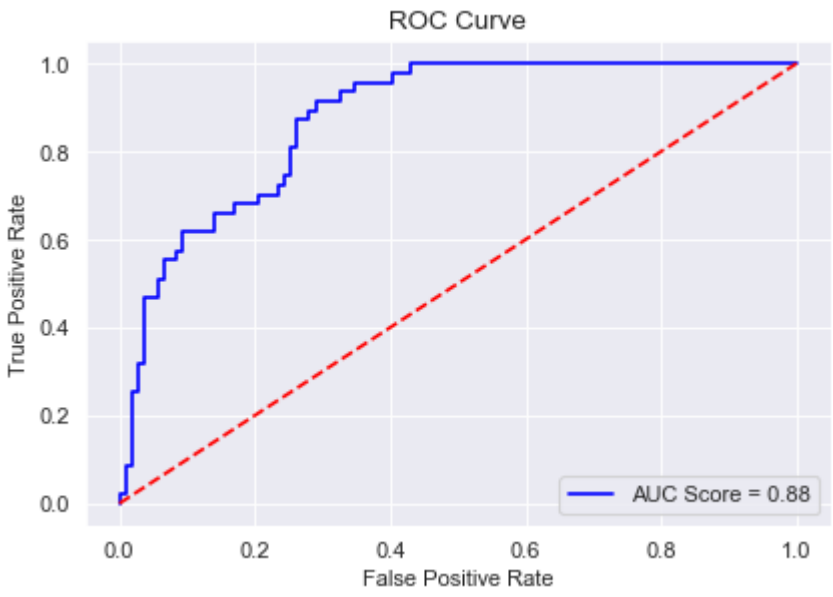
In [126]:
```
from sklearn.svm import SVC
svc_model_linear = SVC(kernel='linear',random_state=0,probability=True,C=0.01)
svc_model_linear.fit(x_train_std,y_train)
svc_pred=svc_model_linear.predict(x_test_std)
```

In [127]:

```
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')

confusion = metrics.confusion_matrix(y_test,svc_pred)
print("Confusion Matrix :""\n", confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

print ("\n", "The confusion matrix explained:")
print("\n","True Positives (TP): we correctly predicted that they do have diabetes: "
,TP)
print("\n","True Negatives (TN): we correctly predicted that they don't have diabete
s:", TN)
print("\n","False Positives (FP): we incorrectly predicted that they do have diabetes
(Type I error)",FP)
print("\n","False Negatives (FN): we incorrectly predicted that they don't have diabe
tes (Type II error)",FN)

# Sensitivity
sensitivity = TP / float(FN + TP)
print("\n","Sensitivity :",sensitivity)

#Specificity
specificity = TN / (TN + FP)
print("\n","Specificity:", specificity)

#  AUC (ROC)
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


Confusion Matrix :
 [[98  9]
 [20 27]]

 The confusion matrix explained:

 True Positives (TP): we correctly predicted that they do have diabetes:  27

 True Negatives (TN): we correctly predicted that they don't have diabetes: 98

 False Positives (FP): we incorrectly predicted that they do have diabetes (Type I e
rror) 9

 False Negatives (FN): we incorrectly predicted that they don't have diabetes (Type
II error) 20

 Sensitivity : 0.574468085106383

 Specificity: 0.9158878504672897

 ROC Curve
```

Out[127]: <matplotlib.legend.Legend at 0x21dec535808>



2.Support Vector Classifier with RBF Kernel

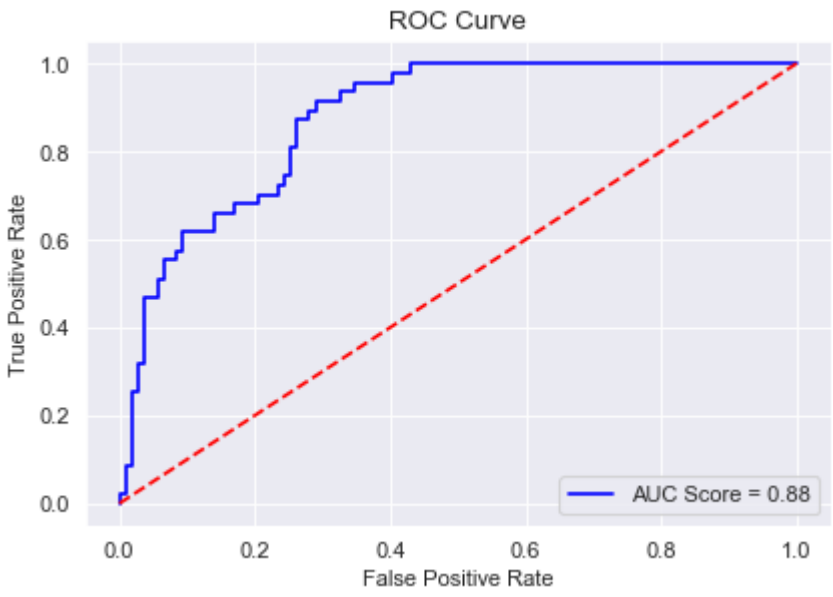
In [128]:
```
svc_model_rbf = SVC(kernel='rbf',random_state=0,probability=True,C=1)
svc_model_rbf.fit(x_train_std,y_train)
svc_pred_rbf=svc_model_rbf.predict(x_test_std)
```

In [129]:

```
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')

confusion = metrics.confusion_matrix(y_test,svc_pred_rbf)
print("Confusion Matrix :""\n", confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

print ("\n", "The confusion matrix explained:")
print("\n","True Positives (TP): we correctly predicted that they do have diabetes: "
,TP)
print("\n","True Negatives (TN): we correctly predicted that they don't have diabete
s:", TN)
print("\n","False Positives (FP): we incorrectly predicted that they do have diabetes
(Type I error)",FP)
print("\n","False Negatives (FN): we incorrectly predicted that they don't have diabe
tes (Type II error)",FN)

# Sensitivity
sensitivity = TP / float(FN + TP)
print("\n","Sensitivity :",sensitivity)

#Specificity
specificity = TN / (TN + FP)
print("\n","Specificity:", specificity)

#  AUC (ROC)
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


Confusion Matrix :
 [[94 13]
 [22 25]]

 The confusion matrix explained:

 True Positives (TP): we correctly predicted that they do have diabetes:  25

 True Negatives (TN): we correctly predicted that they don't have diabetes: 94

 False Positives (FP): we incorrectly predicted that they do have diabetes (Type I e
rror) 13

 False Negatives (FN): we incorrectly predicted that they don't have diabetes (Type
II error) 22

 Sensitivity : 0.5319148936170213

 Specificity: 0.8785046728971962

 ROC Curve
```

Out[129]: <matplotlib.legend.Legend at 0x21dec59cb48>



SVC with Linear Kernel is better than RBF Kernel, This was actually expected beause variables are somewhat depending linearly with outcome

On comparing with KNN both Models are working fine, but SVC Linear is much better in terms of AUC Score.

Logistic Regression

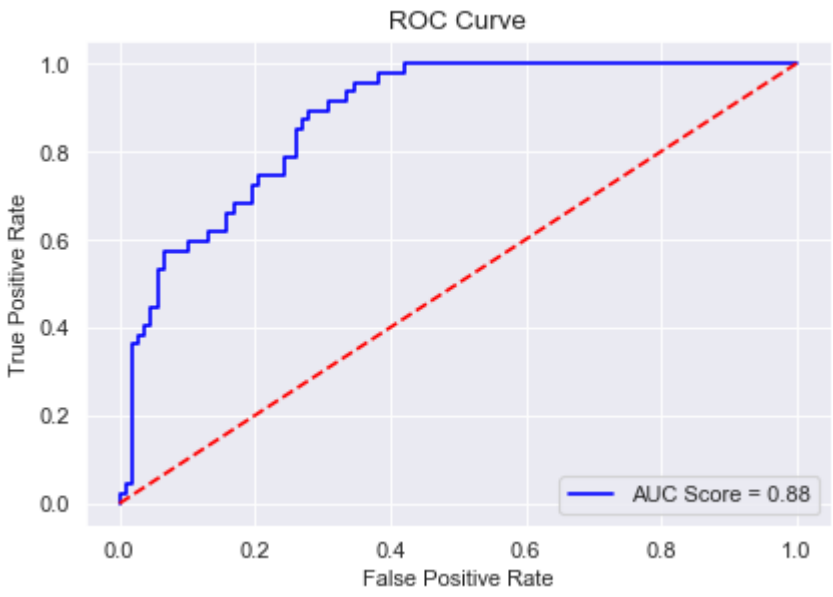
In [130]:
```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(C=0.01)
lr_model.fit(x_train_std,y_train)
lr_pred=lr_model.predict(x_test_std)
```

In [131]:

```
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')

confusion = metrics.confusion_matrix(y_test,lr_pred)
print("Confusion Matrix :""\n", confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

print ("\n", "The confusion matrix explained:")
print("\n","True Positives (TP): we correctly predicted that they do have diabetes: "
,TP)
print("\n","True Negatives (TN): we correctly predicted that they don't have diabete
s:", TN)
print("\n","False Positives (FP): we incorrectly predicted that they do have diabetes
(Type I error)",FP)
print("\n","False Negatives (FN): we incorrectly predicted that they don't have diabe
tes (Type II error)",FN)

# Sensitivity
sensitivity = TP / float(FN + TP)
print("\n","Sensitivity :",sensitivity)

#Specificity
specificity = TN / (TN + FP)
print("\n","Specificity:", specificity)

#  AUC (ROC)
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


Confusion Matrix :
 [[100   7]
 [ 22  25]]

 The confusion matrix explained:

 True Positives (TP): we correctly predicted that they do have diabetes:  25

 True Negatives (TN): we correctly predicted that they don't have diabetes: 100

 False Positives (FP): we incorrectly predicted that they do have diabetes (Type I e
rror) 7

 False Negatives (FN): we incorrectly predicted that they don't have diabetes (Type
II error) 22

 Sensitivity : 0.5319148936170213

 Specificity: 0.9345794392523364

 ROC Curve
```

Out[131]: <matplotlib.legend.Legend at 0x21dec878a48>



We observe that accuracy of KNN is better than that of Logistic Regression, but the AUC score of Logistic Regression is better

Ensemble - Random Forest

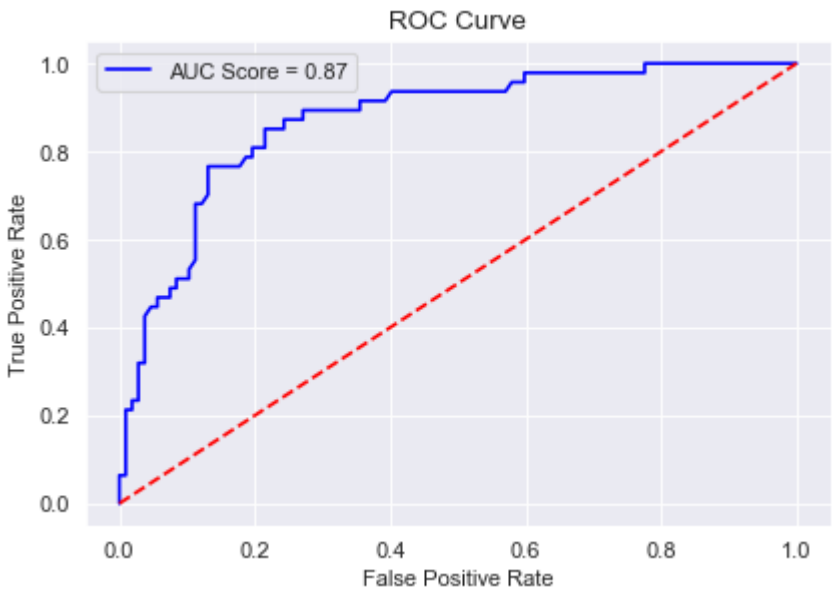
In [132]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=1000,random_state=0)
rf_model.fit(x_train_std,y_train)
rf_pred=rf_model.predict(x_test_std)
```

In [133]:

```
print("Model Validation ==>\n")
print("Accuracy Score of Random Forest Model::")
print(metrics.accuracy_score(y_test,rf_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,rf_pred),'\n')

confusion = metrics.confusion_matrix(y_test,rf_pred)
print("Confusion Matrix :""\n", confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

print ("\n", "The confusion matrix explained:")
print("\n","True Positives (TP): we correctly predicted that they do have diabetes: "
,TP)
print("\n","True Negatives (TN): we correctly predicted that they don't have diabete
s:", TN)
print("\n","False Positives (FP): we incorrectly predicted that they do have diabetes
(Type I error)",FP)
print("\n","False Negatives (FN): we incorrectly predicted that they don't have diabe
tes (Type II error)",FN)

# Sensitivity
sensitivity = TP / float(FN + TP)
print("\n","Sensitivity :",sensitivity)

#Specificity
specificity = TN / (TN + FP)
print("\n","Specificity:", specificity)

#  AUC (ROC)
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 Random Forest 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


Confusion Matrix :
 [[93 14]
 [13 34]]

 The confusion matrix explained:

 True Positives (TP): we correctly predicted that they do have diabetes:  34

 True Negatives (TN): we correctly predicted that they don't have diabetes: 93

 False Positives (FP): we incorrectly predicted that they do have diabetes (Type I e
rror) 14

 False Negatives (FN): we incorrectly predicted that they don't have diabetes (Type
II error) 13

 Sensitivity : 0.723404255319149

 Specificity: 0.8691588785046729

 ROC Curve
```

Out[133]: <matplotlib.legend.Legend at 0x21de5405f08>



So here we observe that Random Forest Classifier is best among all, you might be wondering auc score is lesser by .01 than others We are considering it to be the best because balance of classes between Precision and Recall is far better than other Models, so we can consider a loss in AUC.

Project Task: Week 4

Data Reporting:

Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

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.

Data Reporting inTableau - Creating a Dashboard

Please refer to this link -> https://public.tableau.com/profile/sandip.gujar#!/vizhome/CapstoneProject2Healthcare-week4SandipGujar/HealthcareDashboard?publish=yes (https://public.tableau.com/profile/sandip.gujar#!/vizhome/CapstoneProject2Healthcare-week4SandipGujar/HealthcareDashboard?publish=yes)

END Of PROJECT