In [5]:
```
#week 1 task to upload and clean data with important libraries

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

In [6]:
```
#Uploading the data and EDA

hcd= pd.read_csv("healthcarediabetes.csv")
hcd.head(10)
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	5(
1	1	85	66	29	0	26.6	0.351	3'
2	8	183	64	0	0	23.3	0.672	3
3	1	89	66	23	94	28.1	0.167	2'
4	0	137	40	35	168	43.1	2.288	3
5	5	116	74	0	0	25.6	0.201	3(
6	3	78	50	32	88	31.0	0.248	2(
7	10	115	0	0	0	35.3	0.134	2
8	2	197	70	45	543	30.5	0.158	5
9	8	125	96	0	0	0.0	0.232	5

In [5]:
```
hcd.dtypes
```

Out[5]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [5]:
```
print("Standard Deviation of each variables are ==> ")
hcd.apply(np.std)
```

```
Standard Deviation of each variables are ==>
```

Out[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
```

In [6]:
```
hcd.isnull().any()
```

Out[6]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

In [7]:
```
hcd.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 [8]:
```
Positive = hcd[hcd['Outcome']==1]
Positive.head(10)
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
0	6	148	72	35	0	33.6	0.627	5
2	8	183	64	0	0	23.3	0.672	3
4	0	137	40	35	168	43.1	2.288	3
6	3	78	50	32	88	31.0	0.248	2
8	2	197	70	45	543	30.5	0.158	5
9	8	125	96	0	0	0.0	0.232	5
11	10	168	74	0	0	38.0	0.537	3
13	1	189	60	23	846	30.1	0.398	5
14	5	166	72	19	175	25.8	0.587	5
15	7	100	0	0	0	30.0	0.484	3

In [9]:
```
hcd.describe()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	

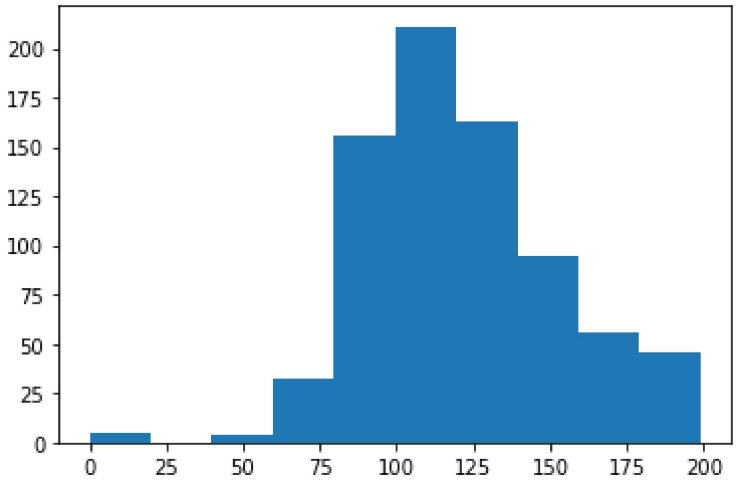
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	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	

In [10]:
```
hcd['Glucose'].value_counts().head(10)
```

Out[10]:
```
100    17
99     17
129    14
125    14
111    14
106    14
95     13
108    13
105    13
102    13
Name: Glucose, dtype: int64
```

In [11]:
```
plt.hist(hcd['Glucose'])
print("Mean of Glucose level is :-", hcd['Glucose'].mean())
print("Datatype of Glucose Variable is:",hcd['Glucose'].dtypes)
```

```
Mean of Glucose level is :- 120.89453125
Datatype of Glucose Variable is: int64
```



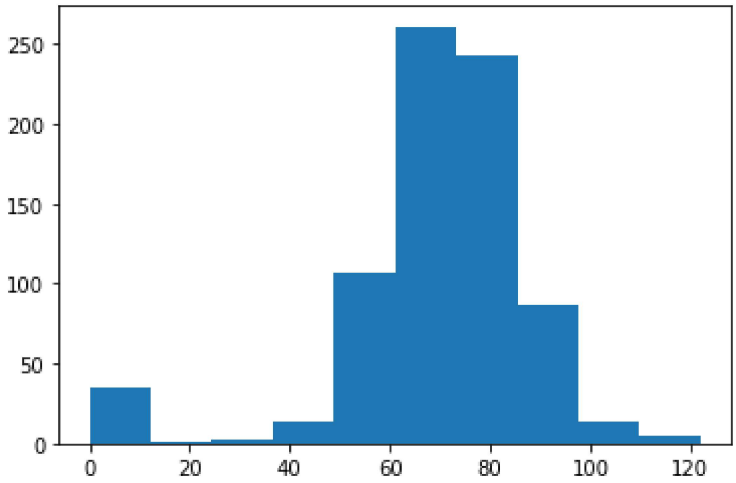
In [12]:
```
hcd['BloodPressure'].value_counts().head(10)
```

Out[12]:
```
70    57
74    52
68    45
78    45
72    44
64    43
80    40
76    39
60    37
```

```
0     35
Name: BloodPressure, dtype: int64
```

In [13]:
```
plt.hist(hcd['BloodPressure'])

print("Mean of Bloodpressure level is :-", hcd['BloodPressure'].mean())
print("Datatype of bloddpressure Variable is:",hcd['BloodPressure'].dtypes)
```

```
Mean of Bloodpressure level is :- 69.10546875
Datatype of bloddpressure Variable is: int64
```

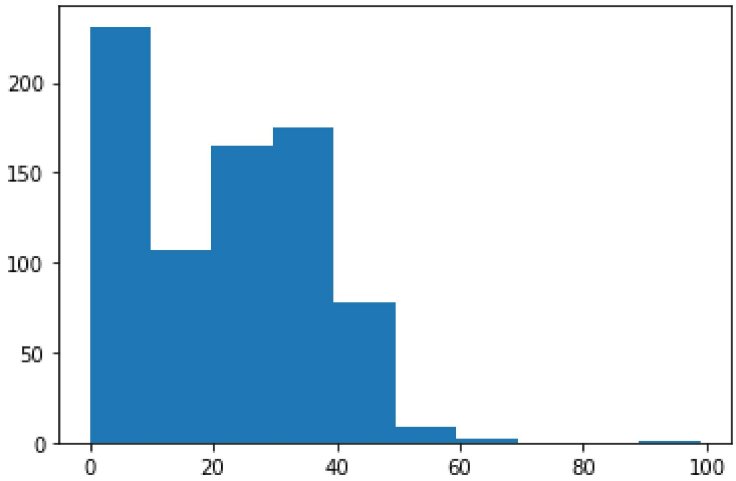


In [14]:
```
hcd['SkinThickness'].value_counts().head(10)
```

Out[14]:
```
0     227
32     31
30     27
27     23
23     22
33     20
18     20
28     20
31     19
39     18
Name: SkinThickness, dtype: int64
```

In [15]:
```
plt.hist(hcd['SkinThickness'])

print("Mean of SkinThicness level is :-", hcd['SkinThickness'].mean())
print("Datatype of SkinTHICKNESS Variable is:",hcd['SkinThickness'].dtypes)
```

```
Mean of SkinThicness level is :- 20.536458333333332
Datatype of SkinTHICKNESS Variable is: int64
```

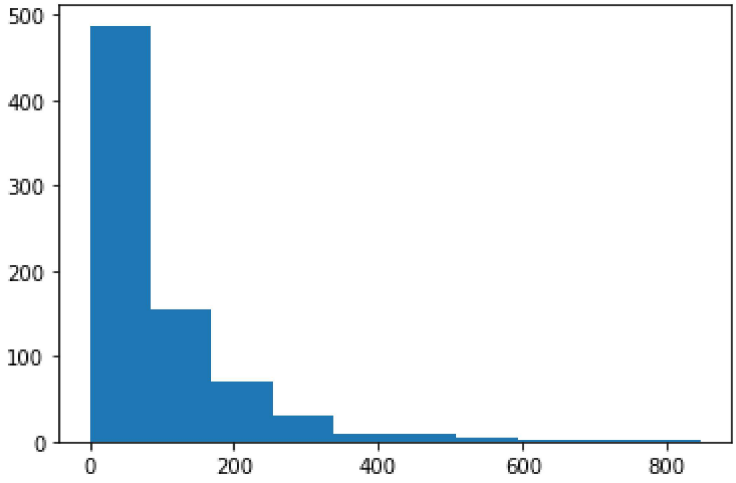


In [16]:
```
hcd['Insulin'].value_counts().head(10)
```

Out[16]:
```
0      374
105     11
140      9
130      9
120      8
100      7
94       7
180      7
110      6
115      6
Name: Insulin, dtype: int64
```

In [17]:
```
plt.hist(hcd['Insulin'])

print("Mean of Insulin level is :-", hcd['Insulin'].mean())
print("Datatype of Insulin Variable is:",hcd['Insulin'].dtypes)
```

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

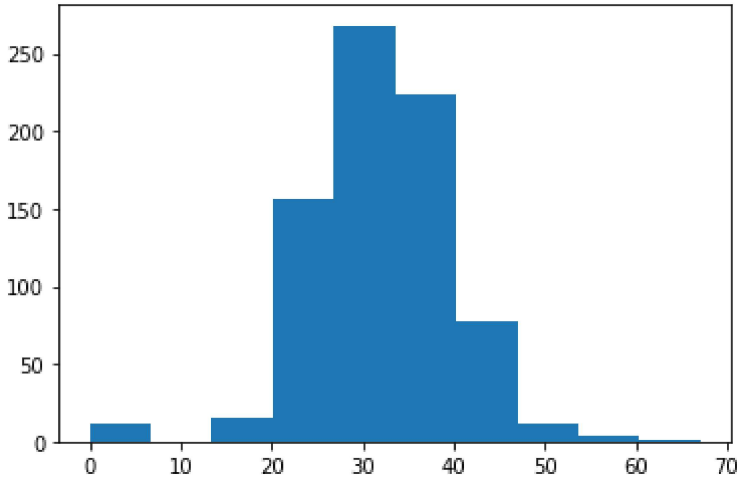


In [18]:
```
hcd['BMI'].value_counts().head(10)
```

Out[18]:
```
32.0    13
31.6    12
31.2    12
0.0     11
33.3    10
32.4    10
32.8     9
30.8     9
32.9     9
30.1     9
Name: BMI, dtype: int64
```

In [19]:
```
plt.hist(hcd['BMI'])
print("Mean of BMI level is :-", hcd['BMI'].mean())
print("Datatype of Glucose Variable is:",hcd['BMI'].dtypes)
```

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



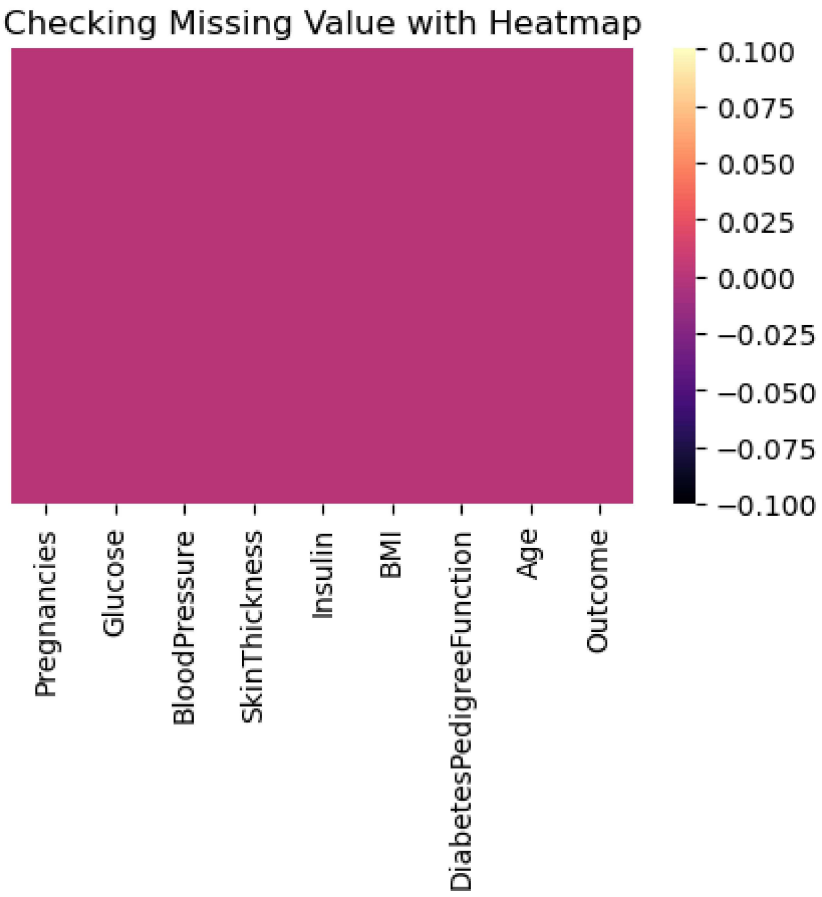
In [20]:
```
hcd.describe().transpose()
```

Out[20]:

	count	mean	std	min	25%	50%	75%	m
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1

In [21]:
```
plt.figure(figsize=(5,3),dpi=100)
plt.title('Checking Missing Value with Heatmap')
sns.heatmap(hcd.isnull(),cmap='magma',yticklabels=False)
```

Out[21]:
```
<AxesSubplot:title={'center':'Checking Missing Value with Heatmap'}>
```

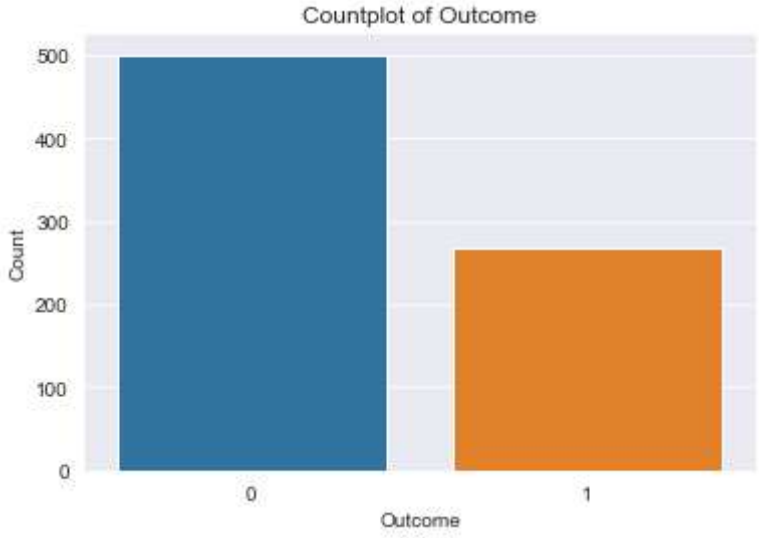


As the above heat map shows that there 0 missing values is the data.

week 2 task Data Exploration

In [22]:
```
sns.set_style('darkgrid')
sns.countplot(hcd['Outcome'])
plt.title("Countplot of Outcome")
plt.xlabel('Outcome')
plt.ylabel("Count")
print("Count of class is:\n",hcd['Outcome'].value_counts())
```

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



In [68]:
```
#scatter plots
```

```
sns.set(style="ticks", color_codes= True)
g= sns.pairplot(hcd,hue="Outcome")
```

We can see from scatter plot that there is no strong multicolinearity among features, but between skin thickness and BMI, Pregnancies and age it looks like there is small chance of positive correlation.

Correlation Analysis

In [24]:
```
hcd.corr()
```

Out[24]:

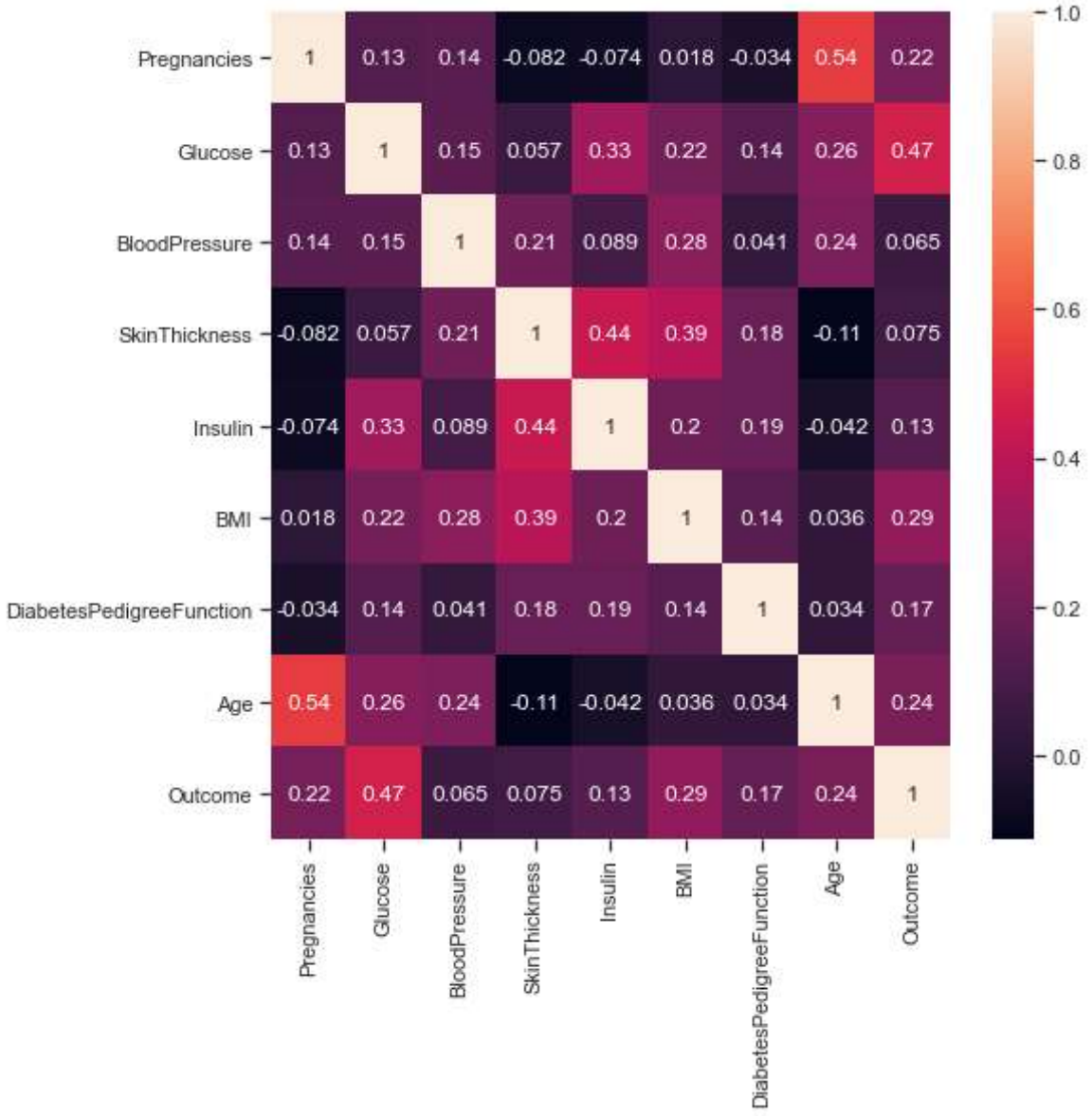
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695

We can clearly see that Glucose and BMI has good impact on outcome. There is a strong positive correlation between BMI(0.29) and Skinthickness(0.074) or Pregnancies(0.22) and age(0.23).

In [70]:
```
plt.figure(dpi=80)
plt.subplots(figsize=(8,8))
sns.heatmap(hcd.corr(),annot= True)
```

Out[70]:
```
<AxesSubplot:>
<Figure size 480x320 with 0 Axes>
```



week 3 task Data Modeling

In [26]:
```
x=hcd.iloc[:,:-1].values
y=hcd.iloc[:,-1].values
```

In [27]:
```
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 [28]:
```
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 [36]:
```
norm=lambda a:(a-min(a))/(max(a)-min(a))
```

In [38]:
```
hcd_norm=hcd.iloc[:,:-1]
```

In [40]:
```
hcd_normalized=hcd_norm.apply(norm)
```

In [41]:
```
x_train_norm,x_test_norm,y_train_norm,y_test_norm=train_test_split(hcd_normalized.va
```

In [42]:
```
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,)
```

KNN with standard scaling

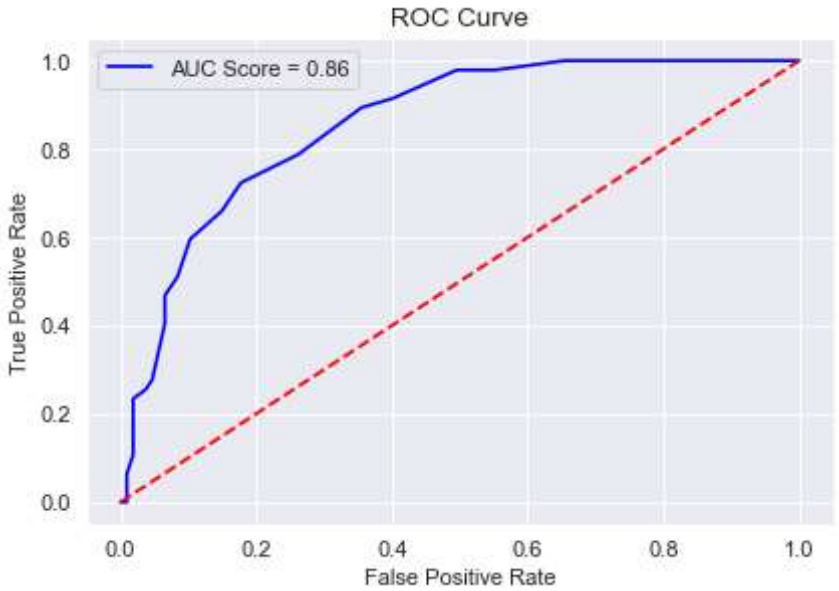
In [43]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=25)
#Using 25 Neighbors just as thumb rule sqrt of observation
knn_model.fit(x_train_std,y_train)
knn_pred=knn_model.predict(x_test_std)
```

In [44]:
```
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[44]: <matplotlib.legend.Legend at 0x15b85f8>



KNN with Normalization.

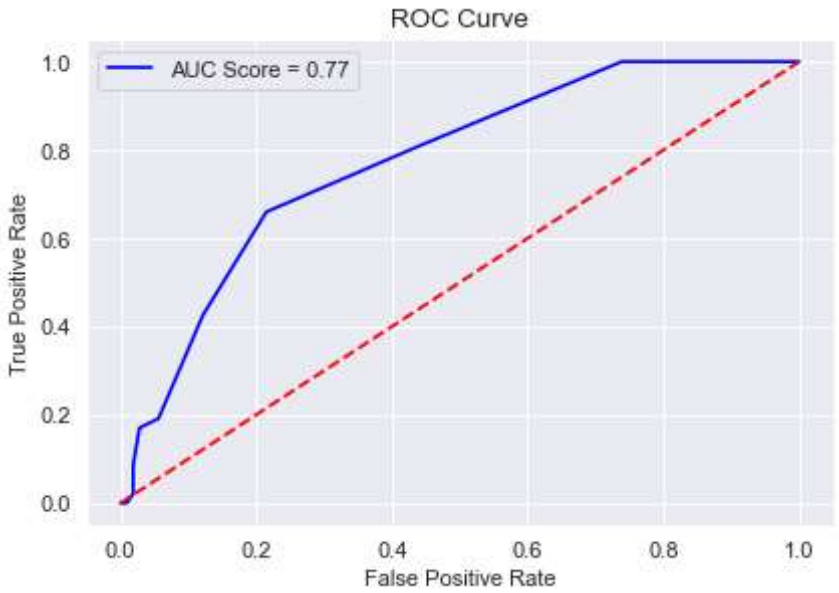
In [45]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model_norm = KNeighborsClassifier(n_neighbors=25)
#Using 25 Neighbors just as thumb rule sqrt of observation
knn_model_norm.fit(x_train_norm,y_train_norm)
knn_pred_norm=knn_model_norm.predict(x_test_norm)
```

In [46]:
```
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[46]: <matplotlib.legend.Legend at 0x161a1f0>



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

In [50]:
```
#standardization

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
hcd_scaled = scaler.fit_transform(hcd)
print(hcd_scaled.mean(axis=0))
print(hcd_scaled.std(axis=0))
```

```
[-6.47630098e-17 -9.25185854e-18  1.50342701e-17  1.00613962e-16
 -3.00685403e-17  2.59052039e-16  2.45174251e-16  1.93132547e-16
  7.40148683e-17]
[1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

As expected, the mean of each variable is now around zero and the standard deviation is set to 1. Thus, all the variable values lie within the same range.

In [51]:
```
#normalization
#Normalization (Min-Max Scalar) :
```

```
#In this approach, the data is scaled to a fixed range – usually 0 to 1.
#In contrast to standardization, the cost of having this bounded range is that we wi
#which can suppress the effect of outliers. Thus MinMax Scalar is sensitive to outli

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
hcd_scaled = scaler.fit_transform(hcd)
```

In [53]:
```
print('means: ', hcd_scaled.mean(axis=0))
print('std: ', hcd_scaled.std(axis=0))
```

```
means:  [0.22617953 0.60751021 0.56643827 0.20743897 0.09432563 0.47678954
 0.16817946 0.20401476 0.34895833]
std:  [0.19808139 0.16056179 0.15855083 0.16102857 0.13613351 0.11742214
 0.14138037 0.19587621 0.47664076]
```

After MinMaxScaling, the distributions are not centered at zero and the standard deviation is not 1.

In [55]:
```
print('Min : ', hcd_scaled.min(axis=0))
print('Max : ', hcd_scaled.max(axis=0))
```

```
Min :  [0. 0. 0. 0. 0. 0. 0. 0. 0.]
Max :  [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

But the minimum and maximum values are standardized across variables, different from what occurs with standardization.

Support Vectore Classifier

In [56]:
```
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)
```

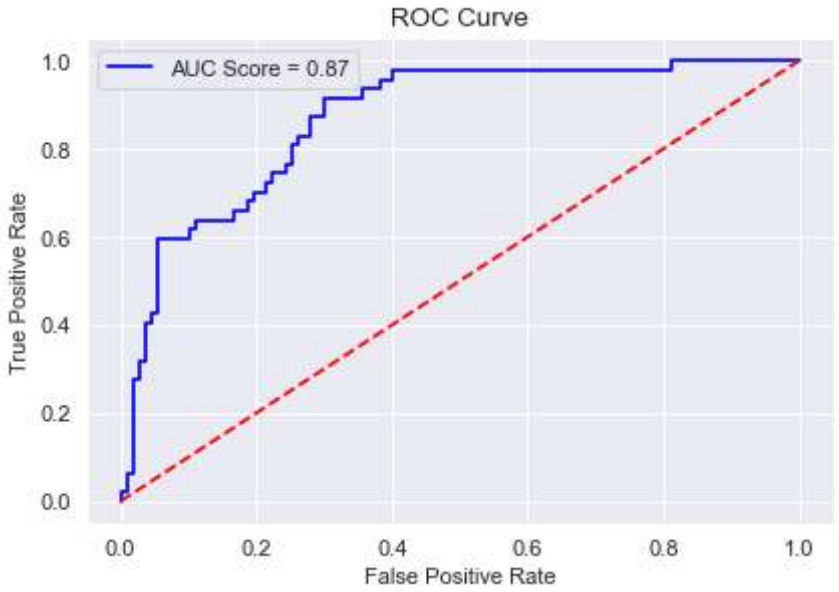
In [57]:
```
print("Model Validation ==>\n")
print("Accuracy Score of SVC Model with Linear Kernel::")
print(metrics.accuracy_score(y_test,svc_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,svc_pred),'\n')
print("\n","ROC Curve")
svc_prob_linear=svc_model_linear.predict_proba(x_test_std)
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[57]: <matplotlib.legend.Legend at 0x1653ad8>



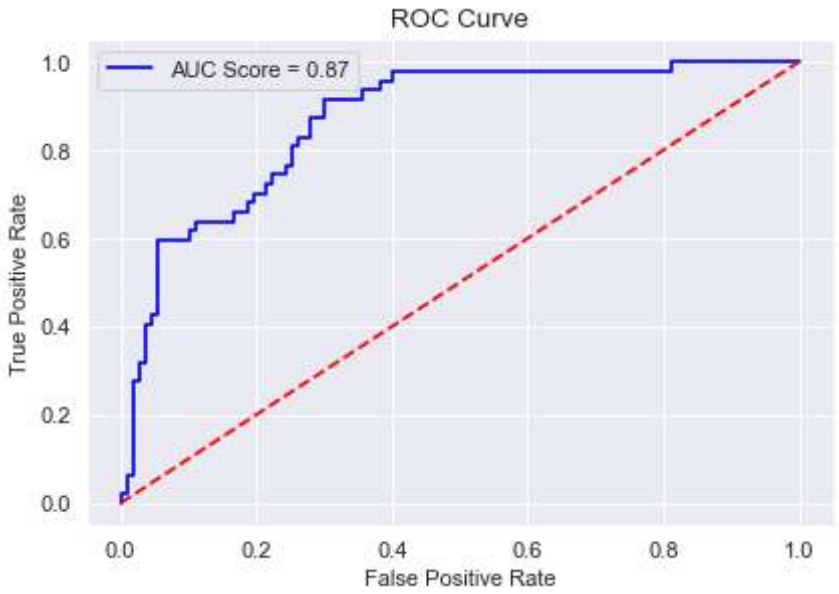
In [58]:
```
from sklearn.svm import SVC
svc_model_rbf = SVC(kernel='rbf',random_state=0,probability=True,C=1)
svc_model_rbf.fit(x_train_std,y_train)
svc_pred_rbf=svc_model_rbf.predict(x_test_std)
```

In [59]:
```
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[59]: <matplotlib.legend.Legend at 0x1745598>



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

Comparing with KNN

Both Models are working fine , but SVC Linear with C=0.01 is better in terms of AUC Score.

Logistic Regression

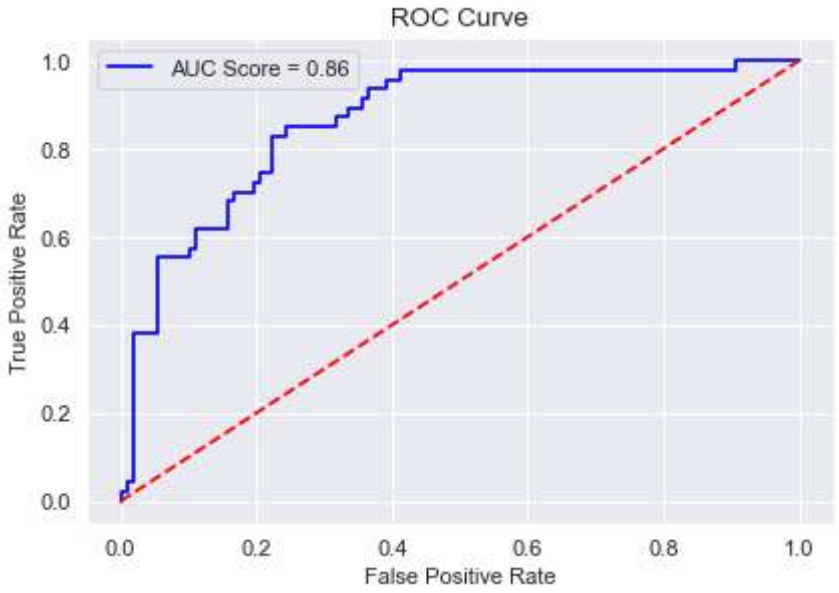
In [60]:
```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(C=0.01)
lr_model.fit(x_train_std,y_train)
lr_pred=lr_model.predict(x_test_std)
```

In [62]:
```
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[62]: <matplotlib.legend.Legend at 0x1716aa8>



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

Ensemble Learning(RF)

In [63]:
```
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)
```

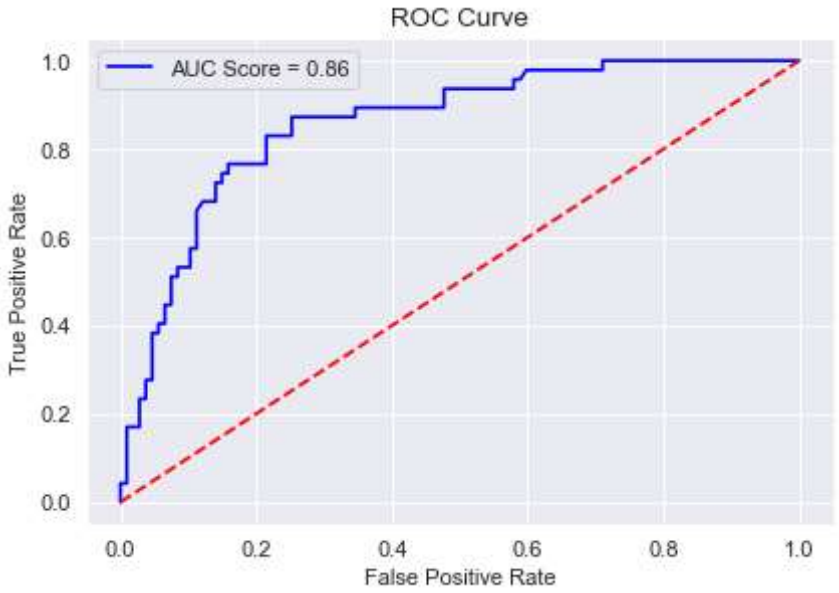
In [64]:
```
print("Model Validation ==>\n")
print("Accuracy Score of Logistic Regression Model::")
print(metrics.accuracy_score(y_test,rf_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,rf_pred),'\n')
print("\n","ROC Curve")
rf_prob=rf_model.predict_proba(x_test_std)
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[64]: <matplotlib.legend.Legend at 0x66f2b38>



So we can see Random Forest Classifier is best among all, you might be wondering auc score is lesser by 1 than others also i am considering it to be best because balance of classes between Precision and Recall is far better than other Models. So we can consider a loss in AUC by 1

In [2]:
```
#creating a report summary
from pandas_profiling import ProfileReport
```

In [7]:
```
profile = ProfileReport(hcd, title="Pandas Profiling Report")
```

In [8]:
```
profile = ProfileReport(hcd, title='Pandas Profiling Report', explorative=True)
```

In [9]:
```
profile
```

Overview

Dataset statistics

Number of variables	9
Number of observations	768
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	54.1 KiB
Average record size in memory	72.1 B

Variable types

NUM	8
BOOL	1

Warnings

`Pregnancies` has 111 (14.5%) zeros	**Zeros**
`BloodPressure` has 35 (4.6%) zeros	**Zeros**
`SkinThickness` has 227 (29.6%) zeros	**Zeros**
`Insulin` has 374 (48.7%) zeros	**Zeros**

Out[9]:

In [ ]:

In [ ]: