

## HEALTHCARE CAPSTONE PROJECT

### DESCRIPTION

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:

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

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

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.

## Data Exploration:

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

```
import numpy as np
import pandas as pd
```

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

```
data = pd.read_csv('health_care_diabetes.csv')
```

```
data.head()
```

	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

```
data.isnull().any()
```

Pregnancies	False
Glucose	False
BloodPressure	False

```

SkinThickness      False
Insulin            False
BMI               False
DiabetesPedigreeFunction  False
Age               False
Outcome           False
dtype: bool

```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 768 entries, 0 to 767
```

```
Data columns (total 9 columns):
```

#	Column	Non-Null Count	Dtype
0	Pregnancies	768 non-null	int64
1	Glucose	768 non-null	int64
2	BloodPressure	768 non-null	int64
3	SkinThickness	768 non-null	int64
4	Insulin	768 non-null	int64
5	BMI	768 non-null	float64
6	DiabetesPedigreeFunction	768 non-null	float64
7	Age	768 non-null	int64
8	Outcome	768 non-null	int64

```
dtypes: float64(2), int64(7)
```

```
memory usage: 54.1 KB
```

```
Positive = data[data['Outcome']==1]
```

```
Positive.head(5)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
BMI \						
0	6	148	72	35	0	33.6
2	8	183	64	0	0	23.3
4	0	137	40	35	168	43.1
6	3	78	50	32	88	31.0
8	2	197	70	45	543	30.5

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
2	0.672	32	1
4	2.288	33	1
6	0.248	26	1
8	0.158	53	1

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

```

100    17
99     17
129    14
125    14
111    14
106    14
95     13
Name: Glucose, dtype: int64

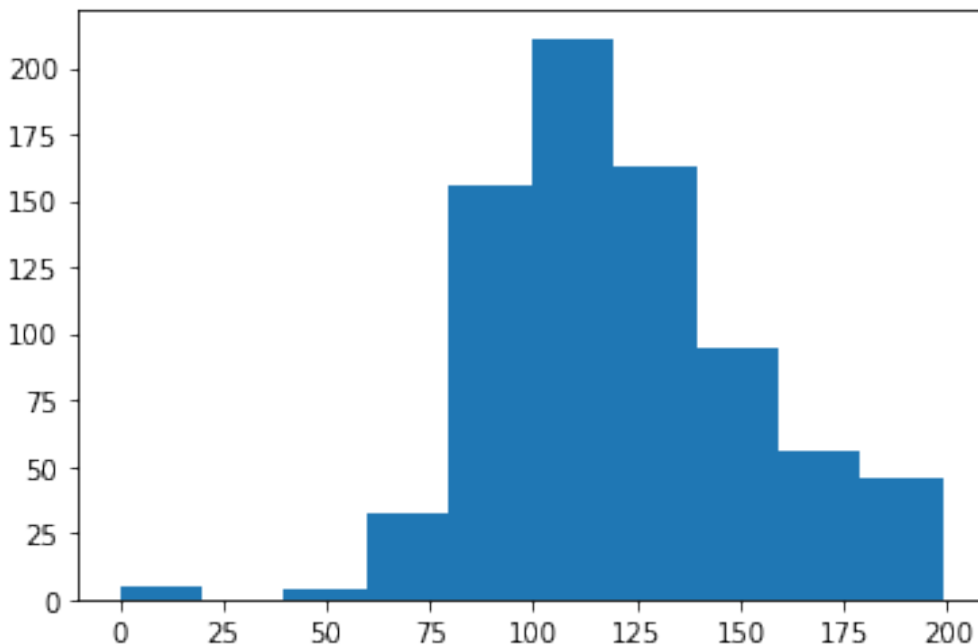
```

```
plt.hist(data['Glucose'])
```

```

(array([ 5.,  0.,  4., 32., 156., 211., 163.,  95.,  56.,  46.]),
 array([ 0., 19.9, 39.8, 59.7, 79.6, 99.5, 119.4, 139.3, 159.2,
        179.1, 199. ]),
 <a list of 10 Patch objects>)

```



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

```

70     57
74     52
68     45
78     45
72     44
64     43
80     40
Name: BloodPressure, dtype: int64

```

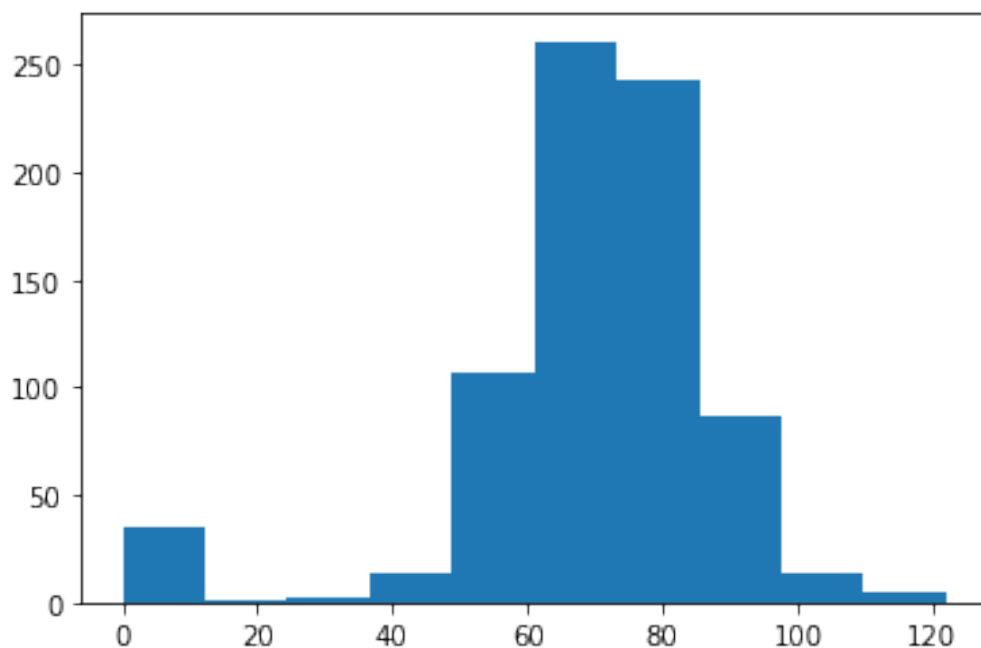
```
plt.hist(data['BloodPressure'])
```

```

(array([ 35.,  1.,  2., 13., 107., 261., 243.,  87., 14.,  5.]),
 array([ 0., 12.2, 24.4, 36.6, 48.8, 61., 73.2, 85.4, 97.6,

```

```
109.8, 122. ]),
<a list of 10 Patch objects>)
```



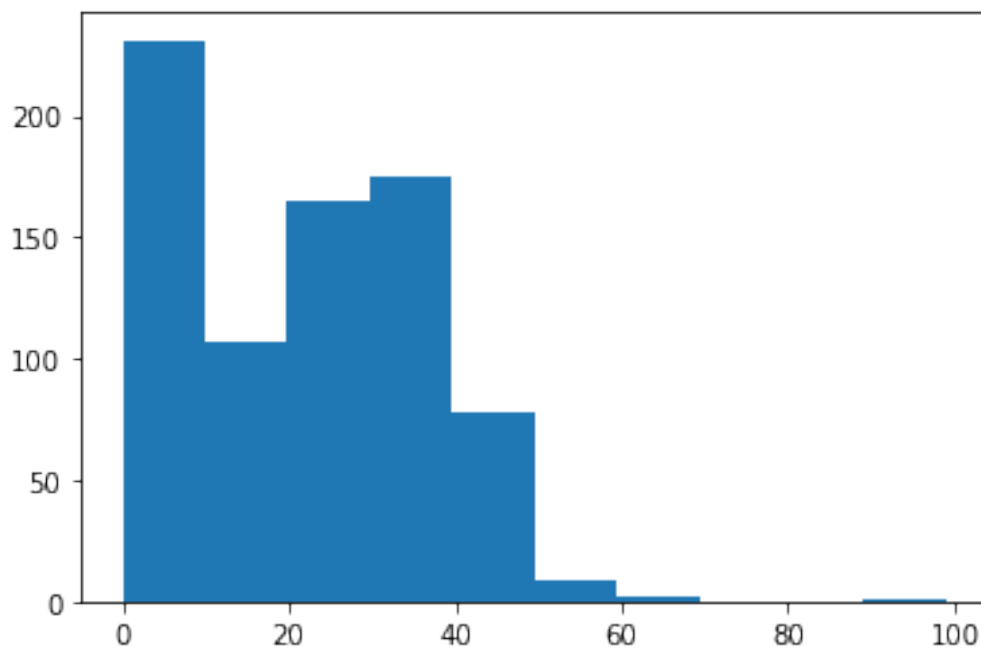
```
data['SkinThickness'].value_counts().head(7)
```

```
0      227
32      31
30      27
27      23
23      22
33      20
18      20
```

```
Name: SkinThickness, dtype: int64
```

```
plt.hist(data['SkinThickness'])
```

```
(array([231., 107., 165., 175., 78., 9., 2., 0., 0., 1.]),
 array([ 0. , 9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1,
99. ]),
<a list of 10 Patch objects>)
```



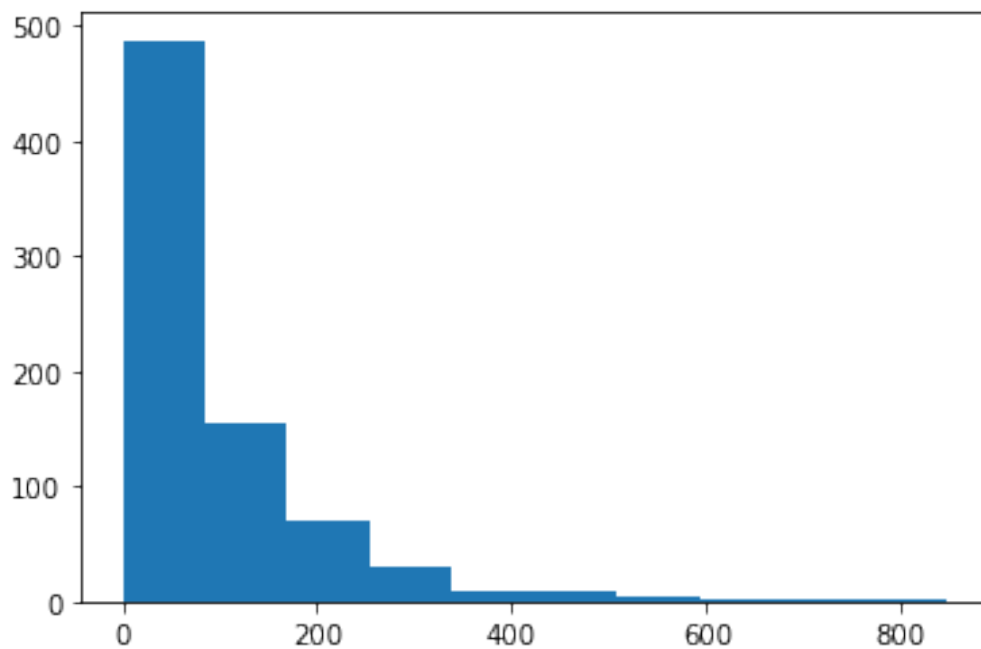
```
data['Insulin'].value_counts().head(7)
```

```
0      374
105     11
140      9
130      9
120      8
100      7
94       7
```

```
Name: Insulin, dtype: int64
```

```
plt.hist(data['Insulin'])
```

```
(array([487., 155., 70., 30., 8., 9., 5., 1., 2., 1.]),
 array([ 0. , 84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,
        761.4, 846. ]),
 <a list of 10 Patch objects>)
```



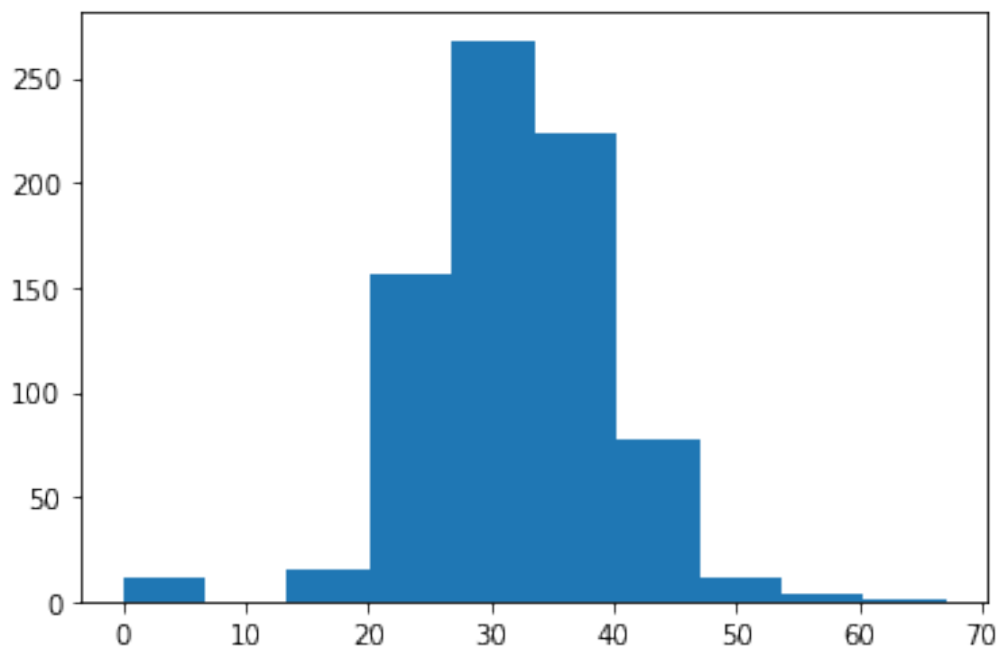
```
data['BMI'].value_counts().head(7)
```

```
32.0    13
31.6    12
31.2    12
0.0     11
33.3    10
32.4    10
32.8     9
```

```
Name: BMI, dtype: int64
```

```
plt.hist(data['BMI'])
```

```
(array([ 11.,   0.,  15., 156., 268., 224.,  78.,  12.,   3.,   1.]),
 array([ 0.,   6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,
        60.39, 67.1 ]),
 <a list of 10 Patch objects>)
```



```
data.describe().transpose()
```

	count	mean	std	min
25% \				
Pregnancies	768.0	3.845052	3.369578	0.000
1.00000				
Glucose	768.0	120.894531	31.972618	0.000
99.00000				
BloodPressure	768.0	69.105469	19.355807	0.000
62.00000				
SkinThickness	768.0	20.536458	15.952218	0.000
0.00000				
Insulin	768.0	79.799479	115.244002	0.000
0.00000				
BMI	768.0	31.992578	7.884160	0.000
27.30000				
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078
0.24375				
Age	768.0	33.240885	11.760232	21.000
24.00000				
Outcome	768.0	0.348958	0.476951	0.000
0.00000				

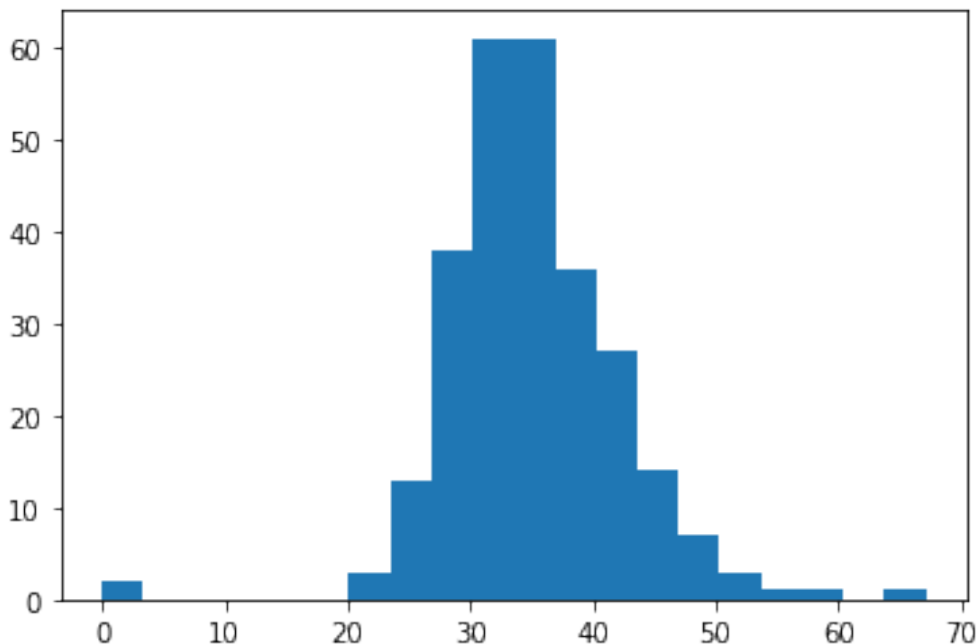
	50%	75%	max
Pregnancies	3.0000	6.00000	17.00
Glucose	117.0000	140.25000	199.00
BloodPressure	72.0000	80.00000	122.00
SkinThickness	23.0000	32.00000	99.00
Insulin	30.5000	127.25000	846.00
BMI	32.0000	36.60000	67.10



DiabetesPedigreeFunction	0.3725	0.62625	2.42
Age	29.0000	41.00000	81.00
Outcome	0.0000	1.00000	1.00

```
plt.hist(Positive['BMI'],histtype='stepfilled',bins=20)

(array([ 2.,  0.,  0.,  0.,  0.,  0.,  3., 13., 38., 61., 61., 36.,
        27.,
         14.,  7.,  3.,  1.,  1.,  0.,  1.]),
 array([ 0.   ,  3.355,  6.71 , 10.065, 13.42 , 16.775, 20.13 ,
        23.485,
         26.84 , 30.195, 33.55 , 36.905, 40.26 , 43.615, 46.97 ,
         50.325,
         53.68 , 57.035, 60.39 , 63.745, 67.1  ]),
 <a list of 1 Patch objects>)
```



```
Positive['BMI'].value_counts().head(7)
```

```
32.9      8
31.6      7
33.3      6
30.5      5
32.0      5
31.2      5
32.4      4
Name: BMI, dtype: int64
```

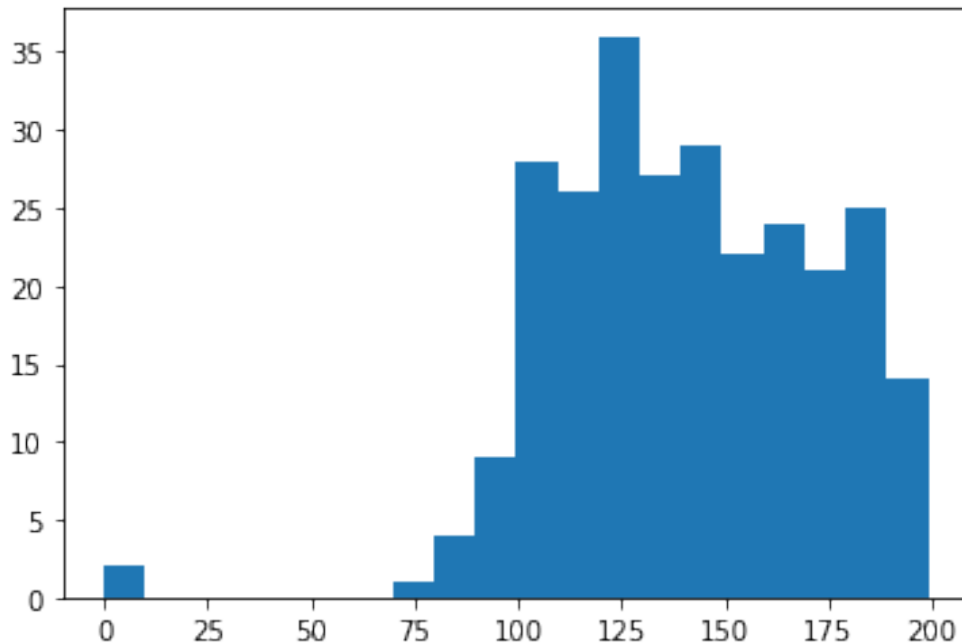
```
plt.hist(Positive['Glucose'],histtype='stepfilled',bins=20)

(array([ 2.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  4.,  9., 28., 26.,
        36.,
```

```

        27., 29., 22., 24., 21., 25., 14.]),
array([ 0. ,  9.95, 19.9 , 29.85, 39.8 , 49.75, 59.7 ,
69.65,
       79.6 , 89.55, 99.5 , 109.45, 119.4 , 129.35, 139.3 ,
149.25,
       159.2 , 169.15, 179.1 , 189.05, 199.  ]),
<a list of 1 Patch objects>)

```



```
Positive['Glucose'].value_counts().head(7)
```

```

125     7
158     6
128     6
115     6
129     6
146     5
162     5
Name: Glucose, dtype: int64

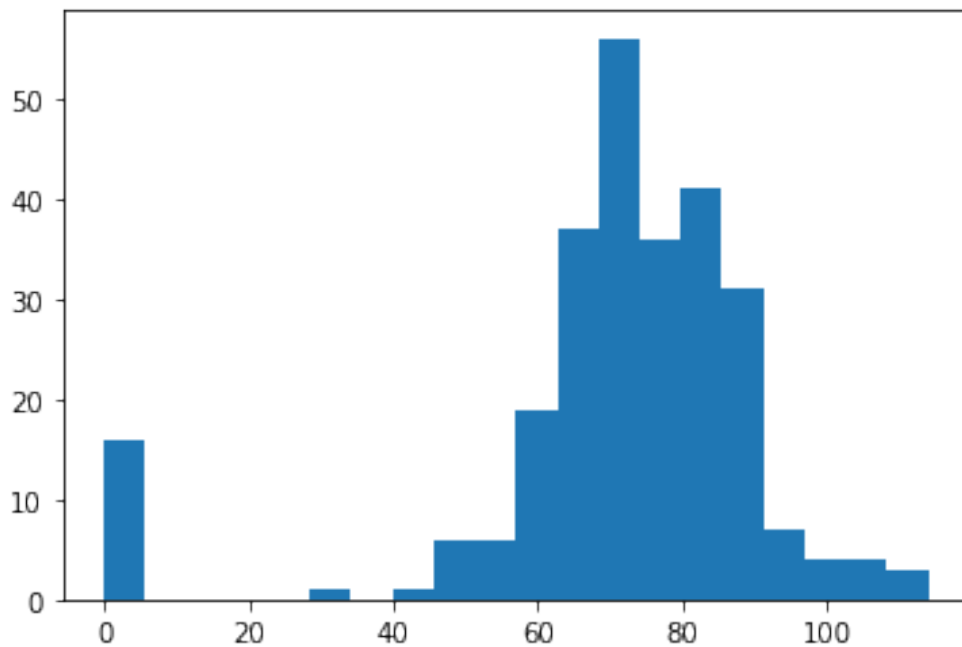
```

```
plt.hist(Positive['BloodPressure'], histtype='stepfilled', bins=20)
```

```

(array([16.,  0.,  0.,  0.,  0.,  1.,  0.,  1.,  6.,  6., 19., 37.,
56.,
       36., 41., 31.,  7.,  4.,  4.,  3.]),
array([ 0. ,  5.7, 11.4, 17.1, 22.8, 28.5, 34.2, 39.9, 45.6,
       51.3, 57. , 62.7, 68.4, 74.1, 79.8, 85.5, 91.2, 96.9,
       102.6, 108.3, 114. ]),
<a list of 1 Patch objects>)

```



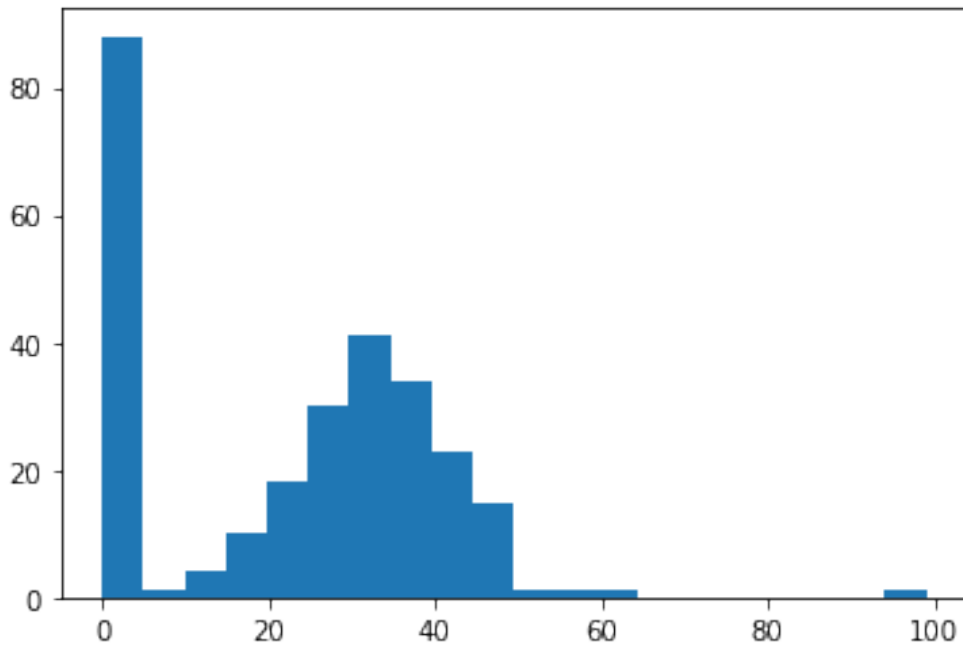
```
Positive['BloodPressure'].value_counts().head(7)
```

```
70    23
76    18
78    17
74    17
72    16
0      16
82    13
```

```
Name: BloodPressure, dtype: int64
```

```
plt.hist(Positive['SkinThickness'],histtype='stepfilled',bins=20)
```

```
(array([88.,  1.,  4., 10., 18., 30., 41., 34., 23., 15.,  1.,  1.,
        1.,
        0.,  0.,  0.,  0.,  0.,  0.,  1.]),
 array([ 0.,  4.95,  9.9 , 14.85, 19.8 , 24.75, 29.7 , 34.65, 39.6 ,
        44.55, 49.5 , 54.45, 59.4 , 64.35, 69.3 , 74.25, 79.2 , 84.15,
        89.1 , 94.05, 99.   ]),
 <a list of 1 Patch objects>)
```



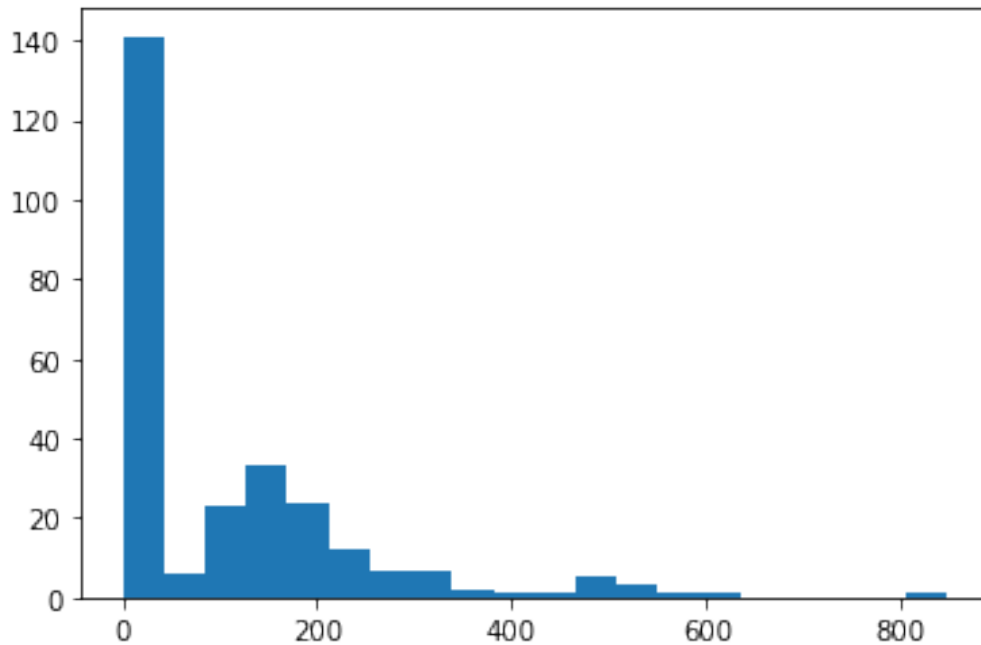
```
Positive['SkinThickness'].value_counts().head(7)
```

```
0      88
32     14
33      9
30      9
39      8
35      8
36      8
```

```
Name: SkinThickness, dtype: int64
```

```
plt.hist(Positive['Insulin'],histtype='stepfilled',bins=20)
```

```
(array([141.,  6., 23., 33., 24., 12.,  7.,  7.,  2.,  1.,
        1.,
           5.,  3.,  1.,  1.,  0.,  0.,  0.,  0.,  1.]),
 array([ 0. , 42.3, 84.6, 126.9, 169.2, 211.5, 253.8, 296.1, 338.4,
        380.7, 423. , 465.3, 507.6, 549.9, 592.2, 634.5, 676.8, 719.1,
        761.4, 803.7, 846. ]),
 <a list of 1 Patch objects>)
```



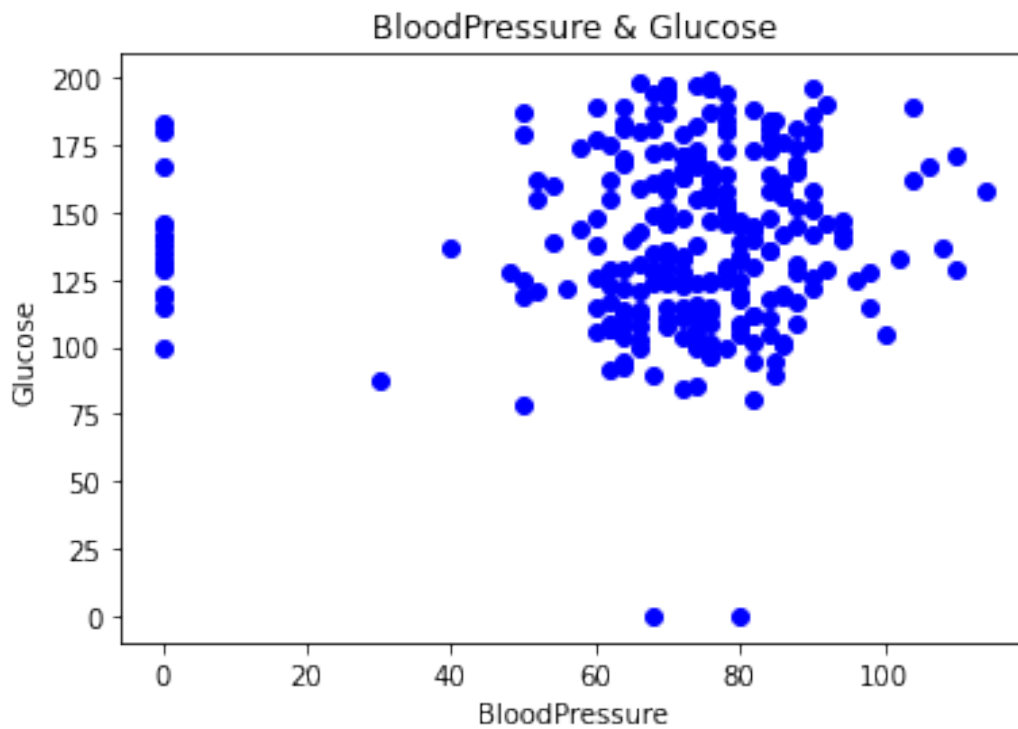
```
Positive['Insulin'].value_counts().head(7)
```

```
0      138
130      6
180      4
156      3
175      3
194      2
125      2
Name: Insulin, dtype: int64
```

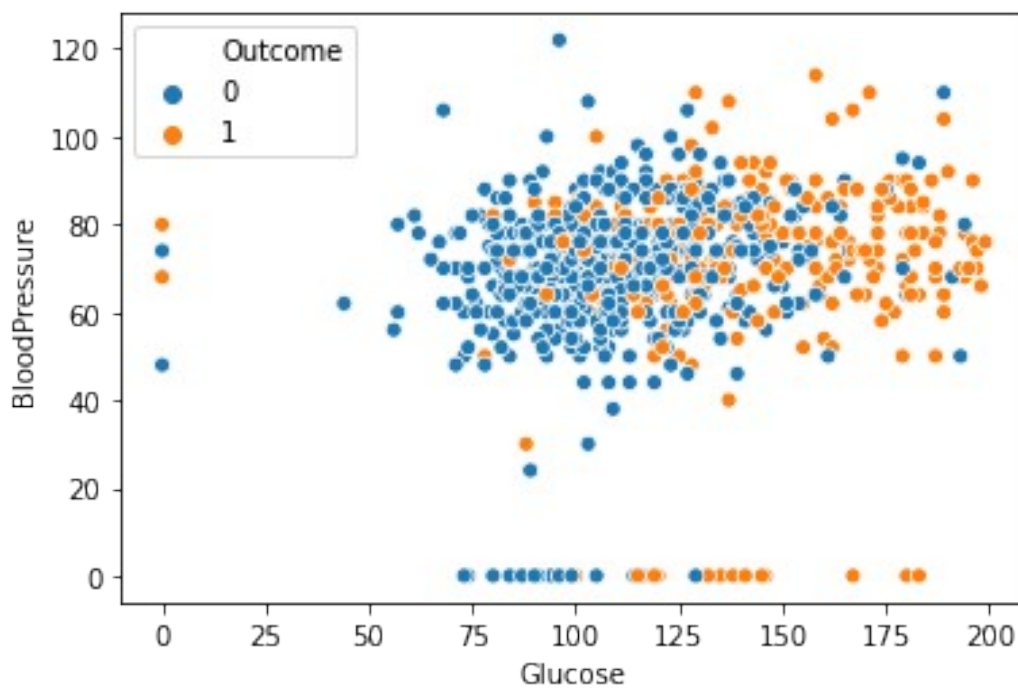
```
#Scatter plot
```

```
BloodPressure = Positive['BloodPressure']
Glucose = Positive['Glucose']
SkinThickness = Positive['SkinThickness']
Insulin = Positive['Insulin']
BMI = Positive['BMI']

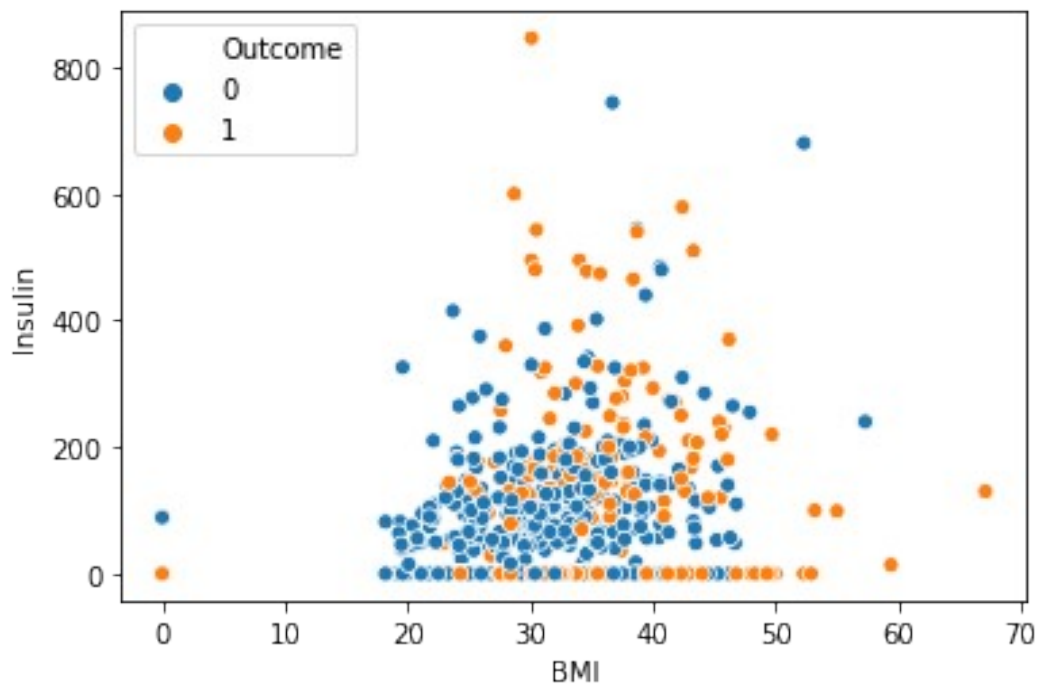
plt.scatter(BloodPressure, Glucose, color=['b'])
plt.xlabel('BloodPressure')
plt.ylabel('Glucose')
plt.title('BloodPressure & Glucose')
plt.show()
```



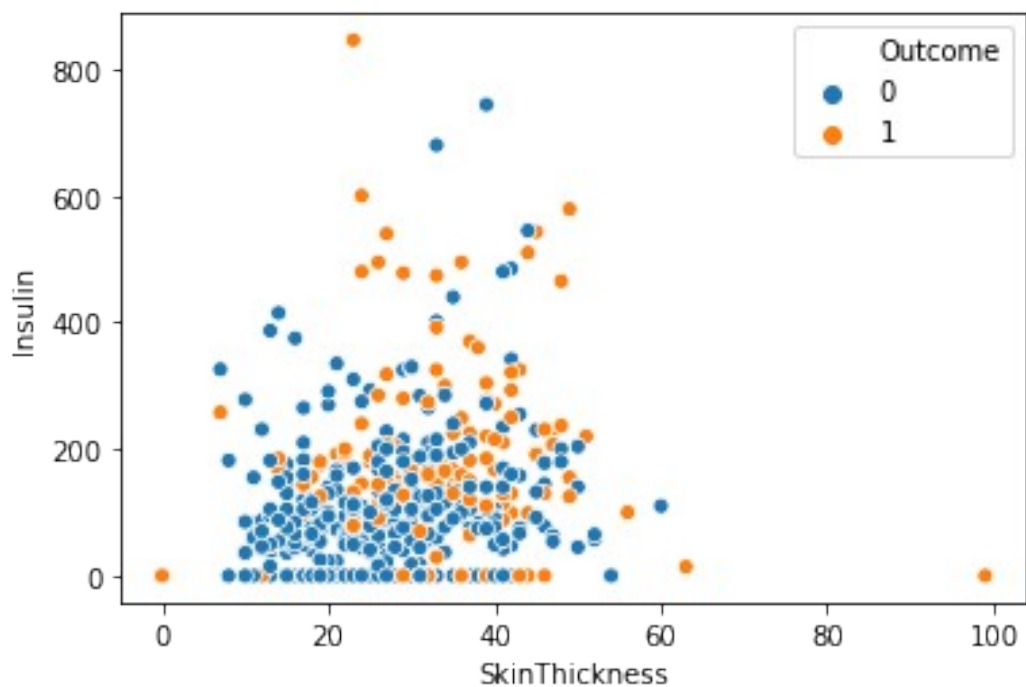
```
g =sns.scatterplot(x= "Glucose" ,y= "BloodPressure",  
                  hue="Outcome",  
                  data=data);
```



```
B =sns.scatterplot(x= "BMI" ,y= "Insulin",  
                  hue="Outcome",  
                  data=data);
```



```
S =sns.scatterplot(x= "SkinThickness" ,y= "Insulin",  
                  hue="Outcome",  
                  data=data);
```



```
### correlation matrix
data.corr()
```

	Pregnancies	Glucose	BloodPressure	
SkinThickness \				
Pregnancies	1.000000	0.129459	0.141282	-
0.081672				
Glucose	0.129459	1.000000	0.152590	
0.057328				
BloodPressure	0.141282	0.152590	1.000000	
0.207371				
SkinThickness	-0.081672	0.057328	0.207371	
1.000000				
Insulin	-0.073535	0.331357	0.088933	
0.436783				
BMI	0.017683	0.221071	0.281805	
0.392573				
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	
0.183928				
Age	0.544341	0.263514	0.239528	-
0.113970				
Outcome	0.221898	0.466581	0.065068	
0.074752				

	Insulin	BMI	DiabetesPedigreeFunction
\			
Pregnancies	-0.073535	0.017683	-0.033523
Glucose	0.331357	0.221071	0.137337
BloodPressure	0.088933	0.281805	0.041265
SkinThickness	0.436783	0.392573	0.183928
Insulin	1.000000	0.197859	0.185071
BMI	0.197859	1.000000	0.140647
DiabetesPedigreeFunction	0.185071	0.140647	1.000000
Age	-0.042163	0.036242	0.033561
Outcome	0.130548	0.292695	0.173844

	Age	Outcome
Pregnancies	0.544341	0.221898
Glucose	0.263514	0.466581
BloodPressure	0.239528	0.065068



```

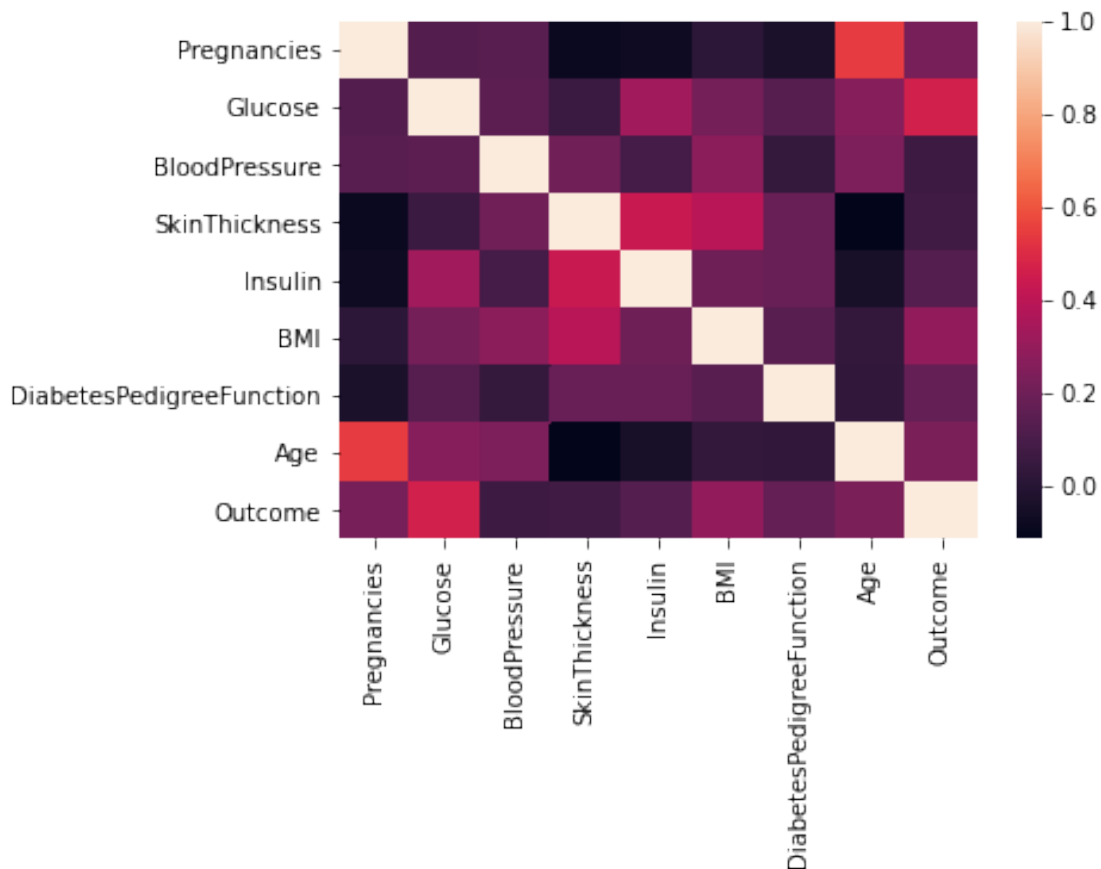
SkinThickness      -0.113970  0.074752
Insulin            -0.042163  0.130548
BMI                0.036242  0.292695
DiabetesPedigreeFunction 0.033561 0.173844
Age                1.000000  0.238356
Outcome            0.238356  1.000000

```

### create correlation heat map

```
sns.heatmap(data.corr())
```

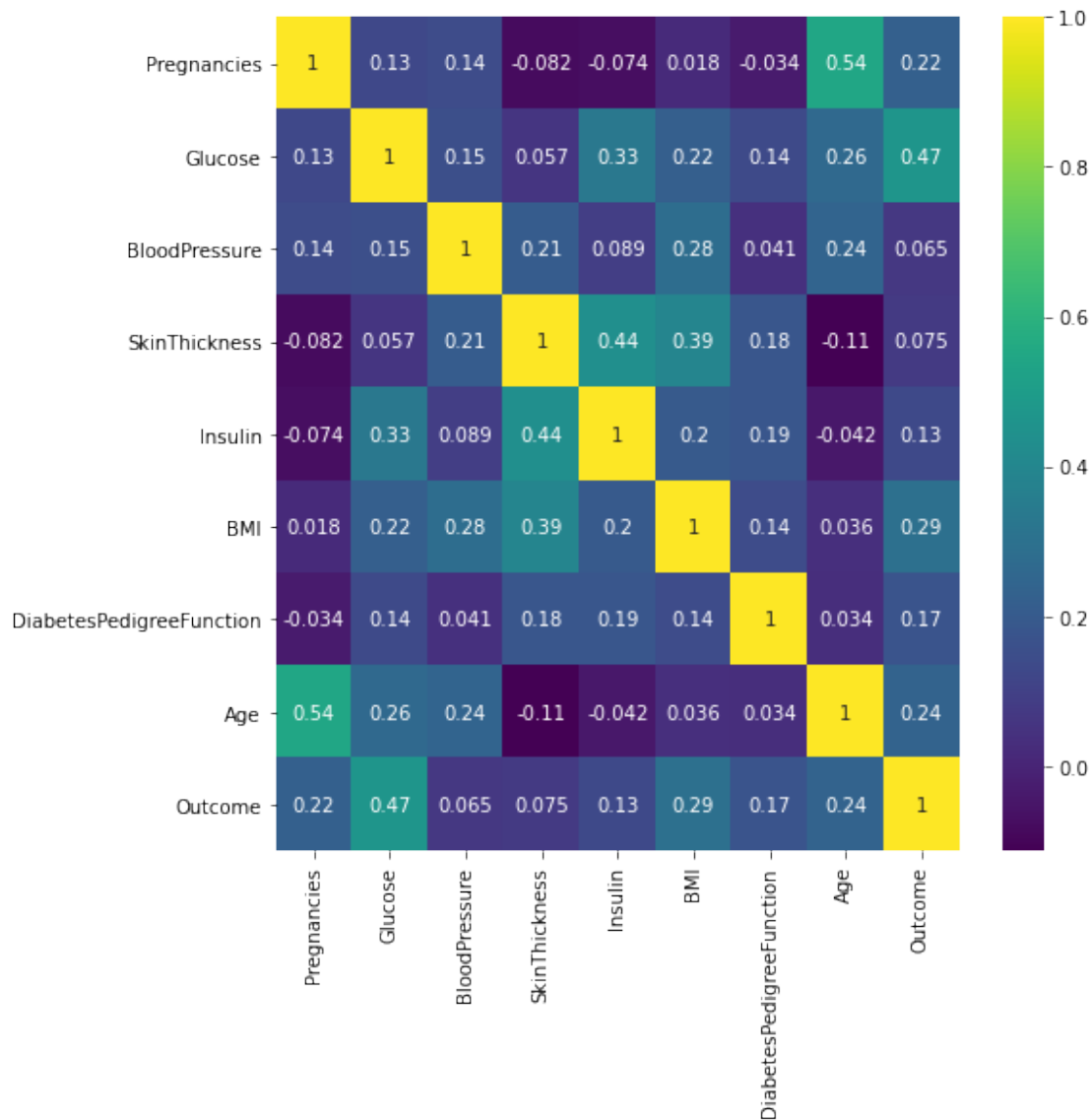
<matplotlib.axes.\_subplots.AxesSubplot at 0x7fe4d495d520>



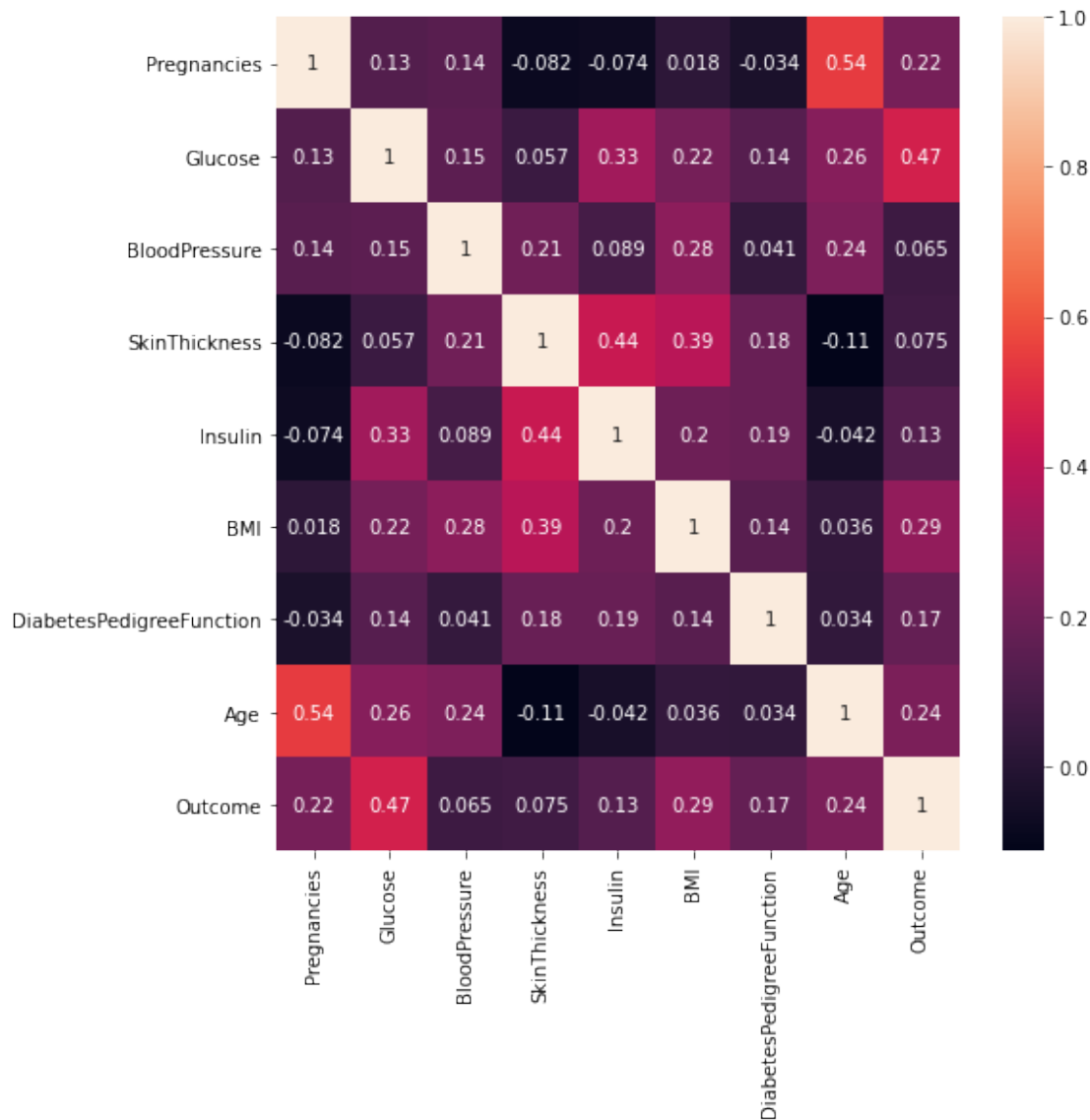
```
plt.subplots(figsize=(8,8))
```

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

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fe4d4a6bc10>



```
plt.subplots(figsize=(8,8))
sns.heatmap(data.corr(),annot=True)  ### gives correlation value
<matplotlib.axes._subplots.AxesSubplot at 0x7fe4d4e2a340>
```



## Project Task Week 2

### Data Modeling:

- Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.
- Apply an appropriate classification algorithm to build a model.
- Compare various models with the results from KNN algorithm.
- 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.

```

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

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=0)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

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

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))

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

data_normalized=data_norm.apply(norm)

x_train_norm,x_test_norm,y_train_norm,y_test_norm=train_test_split(data_normalized.values,y,test_size=0.20,random_state=0)

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,)

```

Since the data are numerical, the most effective method is Logistic Regression. Additionally, all variables are dependent on every target linearly this also favorable for logistics regression.

Note that this learning approach also works well with linear data. To validate the model, I will use a train-test split; for accuracy, I will use the confusion matrix because the classes are balanced; and I will also evaluate the ROC Curve and ROC AUC Score to ensure that Type 2 Error does not arise for the Positive class, which is number 1.

### KNN with standard scaling

```
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)

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 ==>

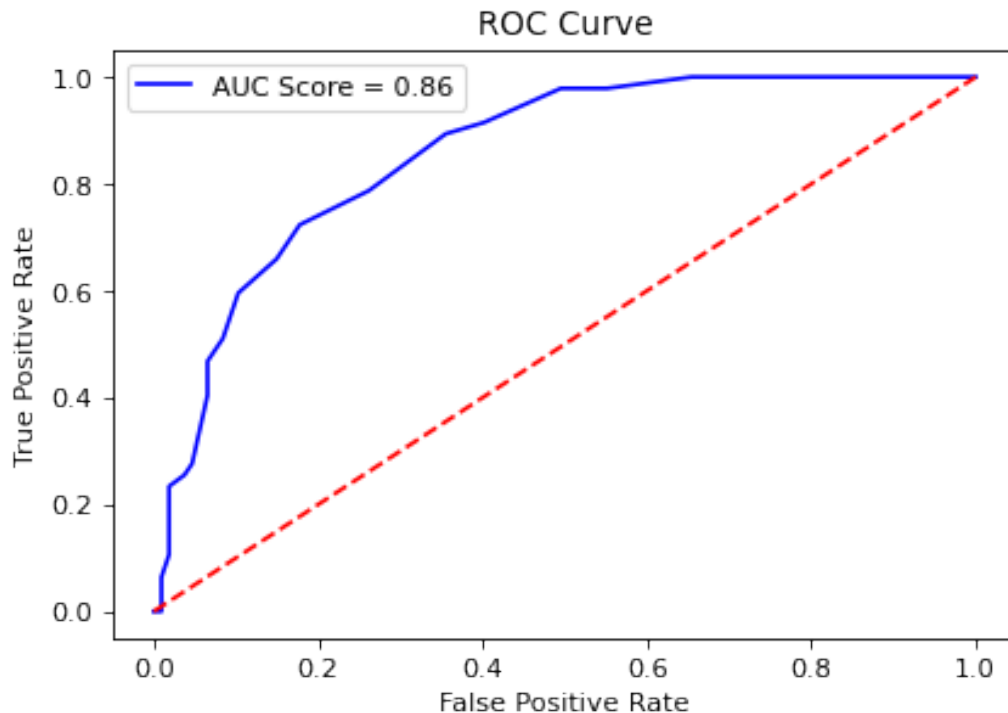
Accuracy Score of KNN Model::  
0.7922077922077922

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

<matplotlib.legend.Legend at 0x7fe4d41c7dc0>



### KNN With Normalization

```
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)

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

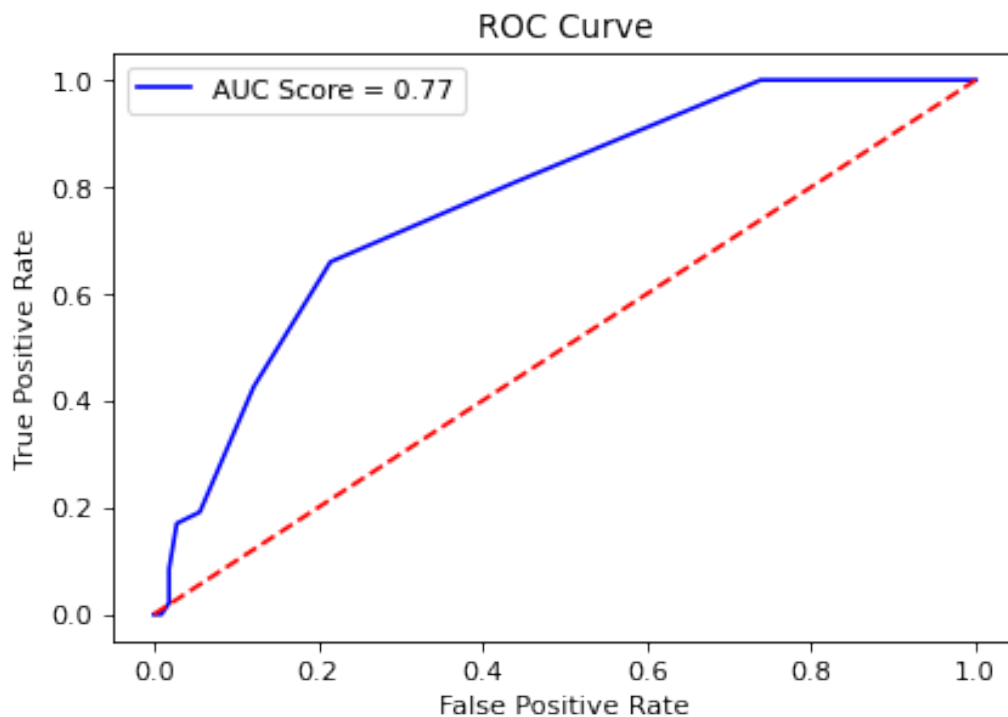
```
Classification Report::
              precision    recall  f1-score   support

     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

<matplotlib.legend.Legend at 0x7fe4d51e1f40>



It is clear that KNN with Standardization is better than Normalization so I built models using Z Score Standardization and will compare with KNN

### Support Vector Classifier

```
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)
```

```

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

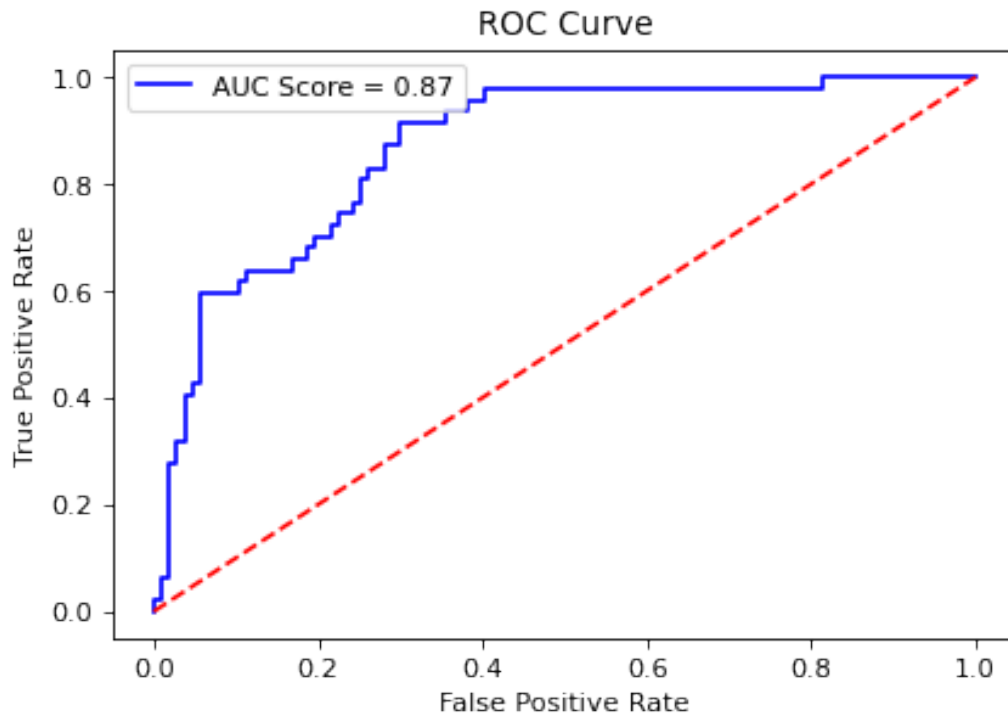
Classification Report::

	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

<matplotlib.legend.Legend at 0x7fe4d4481fd0>





```
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)

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 ==>

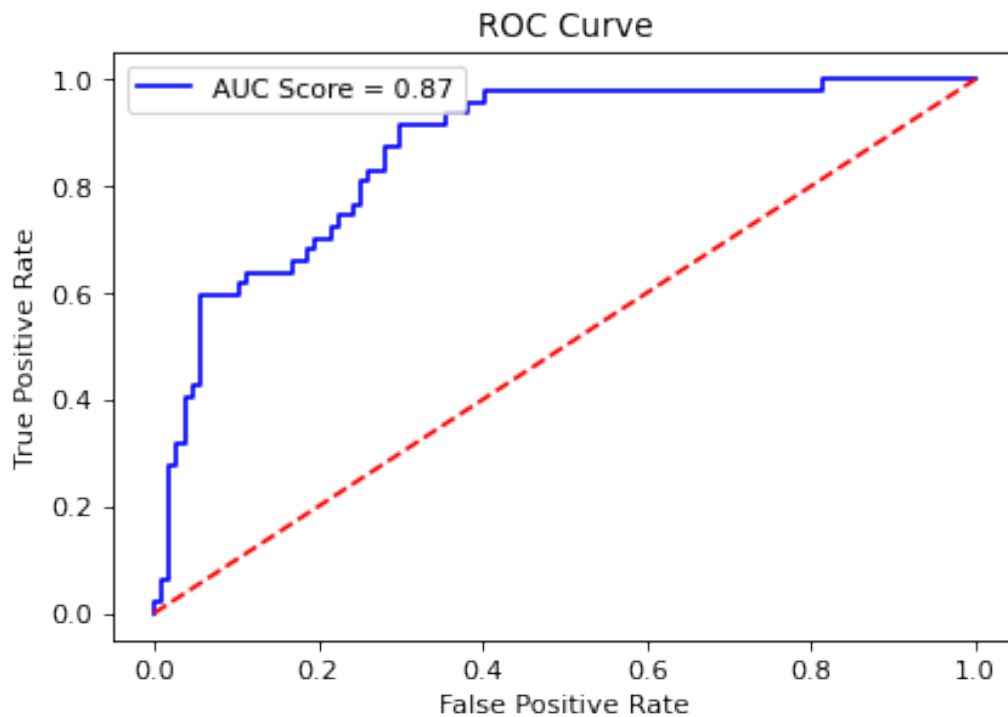
Accuracy Score of SVC Model with RBF Kernel::  
0.7922077922077922

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

<matplotlib.legend.Legend at 0x7fe4d564ef40>



SVC with Linear Kernel is better than RBF Kernel as I compared it since the variables are 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

```
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)
```

```

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()

```

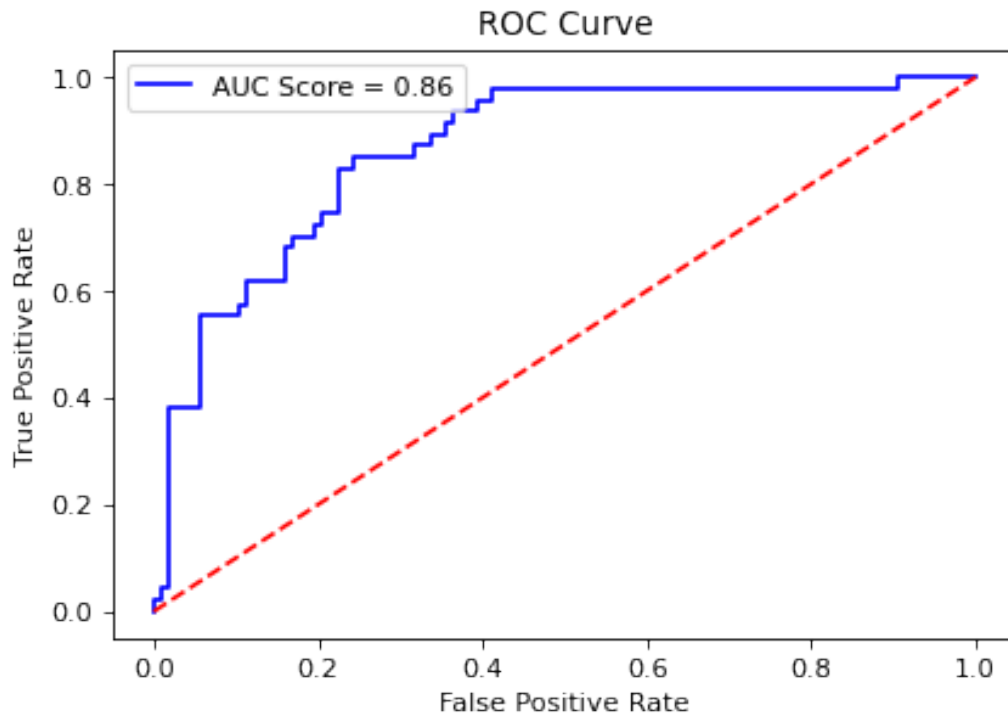
Model Validation ==>

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

<matplotlib.legend.Legend at 0x7fe4d57309a0>



Accuracy of KNN is better than Logistic Regression, but AUC Score of Logistic regression is better according to the model.

### Ensemble Learning (Random Forest)

```
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)

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 ==>

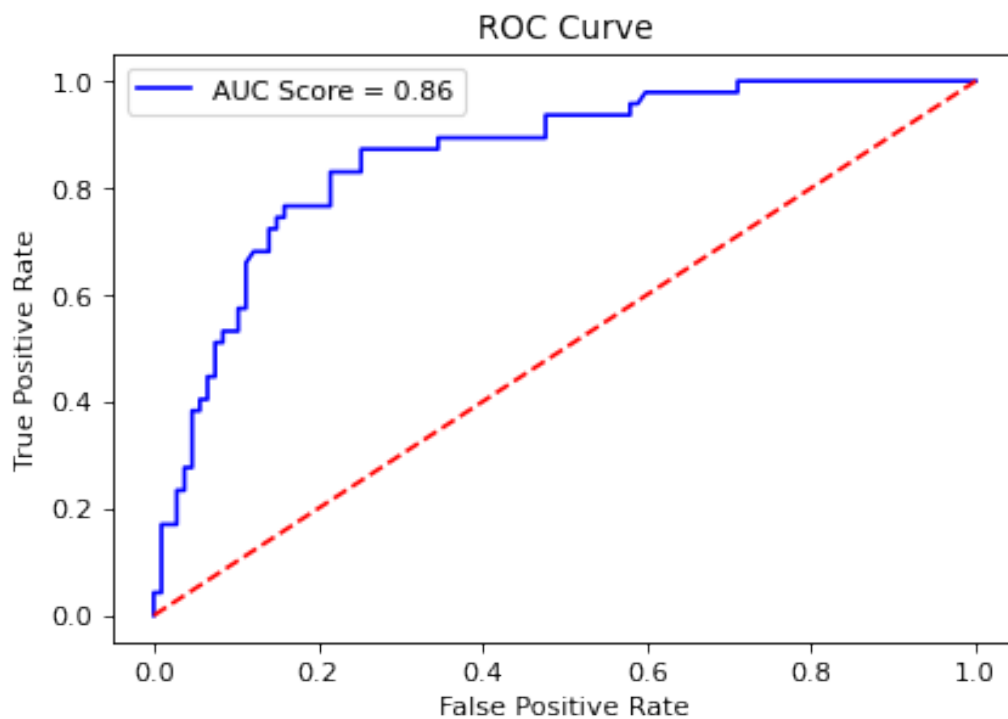
Accuracy Score of Logistic Regression Model::  
0.81818181818182

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

<matplotlib.legend.Legend at 0x7fe4b77c8880>



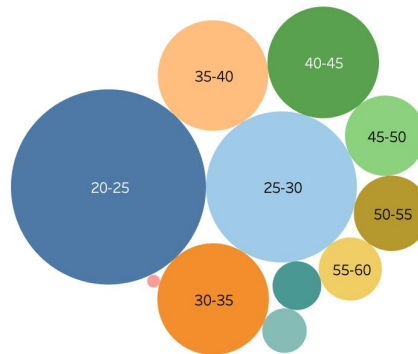
Random Forest Classifier is the best of the models even though the AUC score is one point lower than the others, it still remain balance between precision and recall which considerably superior to other models.

Outcome  
0  
1

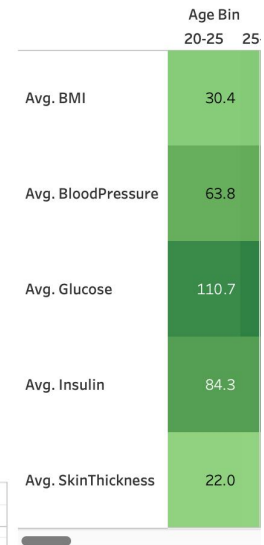
Pie Chart



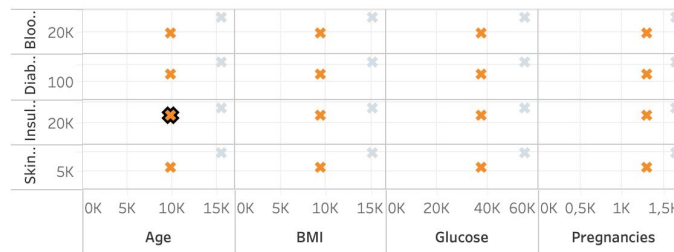
Bubble Chart



Heat Map



Scatter plot



Histogram Plot

