In [3]:

```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
#from sklearn.feature_extraction.text import CountVectorizer  #DT does not take strings
from IPython.display import Image
#import pydotplus as pydot
from sklearn import tree
from os import system
```

In [4]:

```
pdata = pd.read_csv("diabetes.csv")
```

In [5]:

```
pdata.shape
```

Out[5]:

```
(768, 9)
```

In [6]:

```
pdata.head()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.

In [7]:

```
pdata.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]:

```
pdata.dtypes
```

Out[8]:

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [9]:

```
pdata.isnull().values.any()
```

Out[9]:

```
False
```

In [11]:

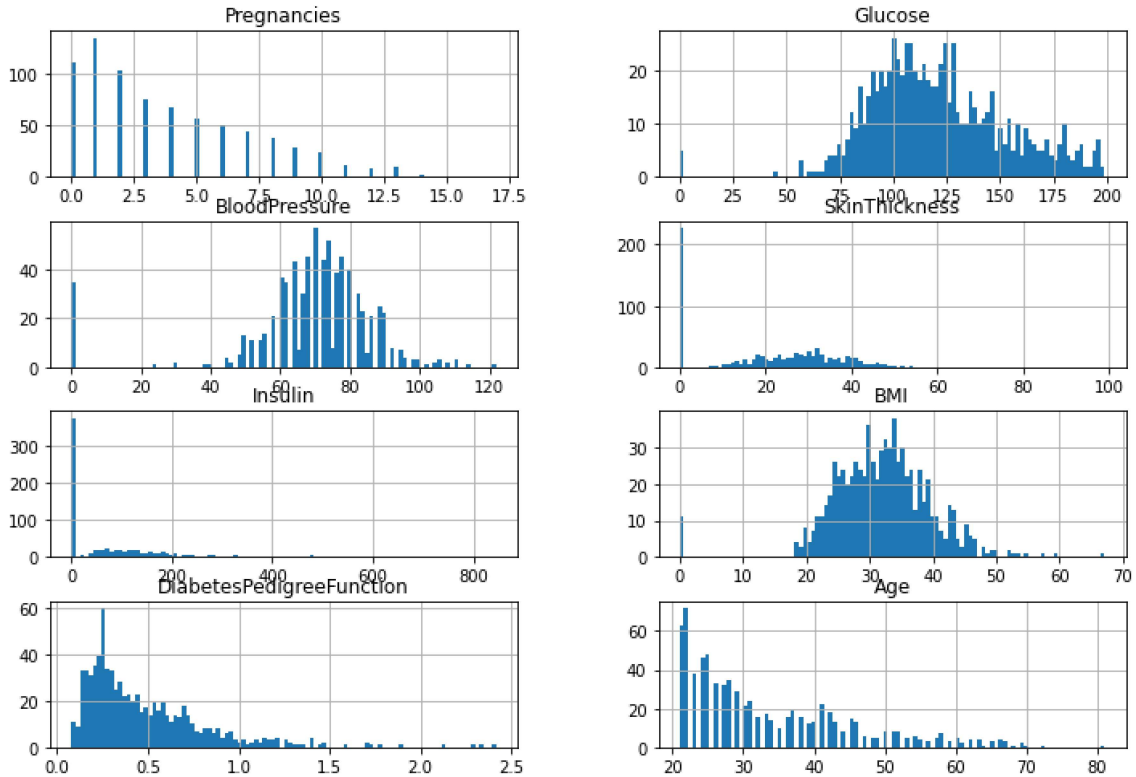
```
pdata.describe(include='all')
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diab
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
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]:

```
columns = list(pdata)[0:-1] # Excluding Outcome column which has only
pdata[columns].hist(stacked=False, bins=100, figsize=(12,30), layout=(14,2));
```

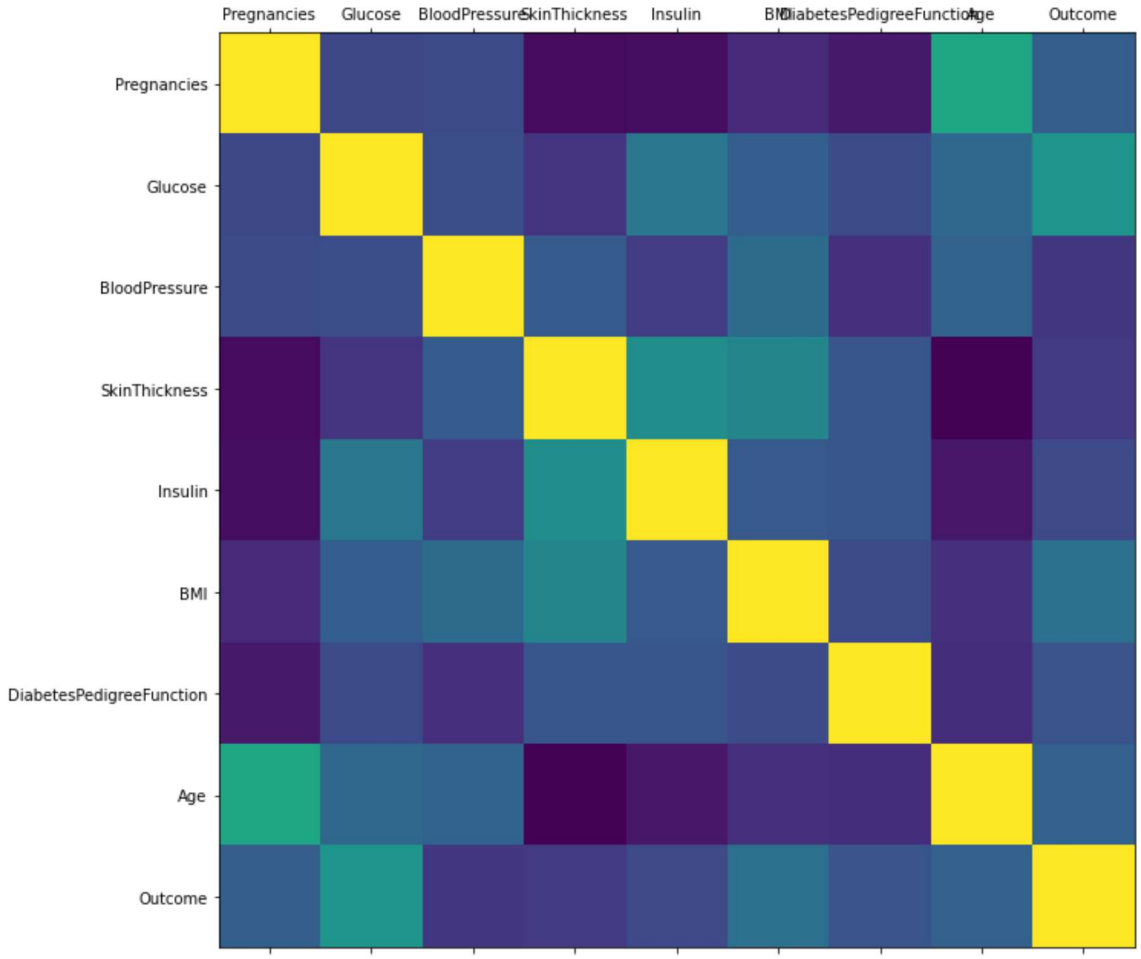


In [12]:

```
def plot_corr(df, size=11):
    corr = df.corr()
    fig, ax = plt.subplots(figsize=(size, size))
    ax.matshow(corr)
    plt.xticks(range(len(corr.columns)), corr.columns)
    plt.yticks(range(len(corr.columns)), corr.columns)
```

In [13]:

```
plot_corr(pdata)
```



In [12]:

```
X = pdata.drop("Outcome" , axis=1)
y = pdata.pop("Outcome")
```

In [16]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30, random_state=1)
```

In [17]:

```
dTree = DecisionTreeClassifier(criterion = 'gini', random_state=1)
dTree.fit(X_train, y_train)
```

Out[17]:

```
DecisionTreeClassifier(random_state=1)
```

In [18]:

```
print(dTree.score(X_train, y_train))
print(dTree.score(X_test, y_test))
```

```
1.0
0.6883116883116883
```

In [19]:

```
train_char_label = ['No', 'Yes']
Credit_Tree_File = open('credit_tree.dot','w')
dot_data = tree.export_graphviz(dTree, out_file=Credit_Tree_File, feature_names = list(X
Credit_Tree_File.close()
```

In [21]:

```
dTreeR = DecisionTreeClassifier(criterion = 'gini', max_depth = 3, random_state=1)
dTreeR.fit(X_train, y_train)
print(dTreeR.score(X_train, y_train))
print(dTreeR.score(X_test, y_test))
```

```
0.7635009310986964
0.7575757575757576
```

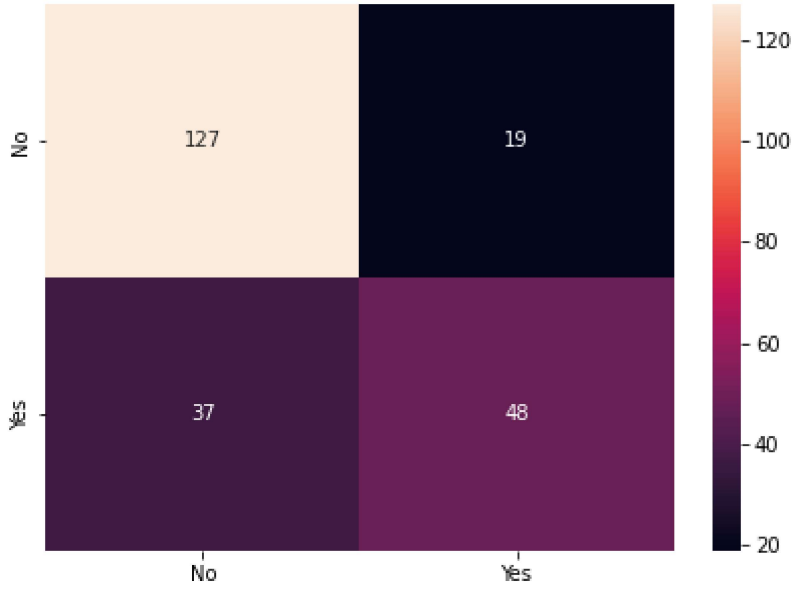
In [22]:

```
print(dTreeR.score(X_test , y_test))
y_predict = dTreeR.predict(X_test)

cm=metrics.confusion_matrix(y_test, y_predict, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["No","Yes"]],
                  columns = [i for i in ["No","Yes"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
```

```
0.7575757575757576
```

Out[22]:

```
<AxesSubplot:>
```



In [23]:

```
from sklearn.ensemble import BaggingClassifier

bgcl = BaggingClassifier(base_estimator=dTree, n_estimators=50,random_state=1)
#bgcl = BaggingClassifier(n_estimators=50,random_state=1)

bgcl = bgcl.fit(X_train, y_train)
```

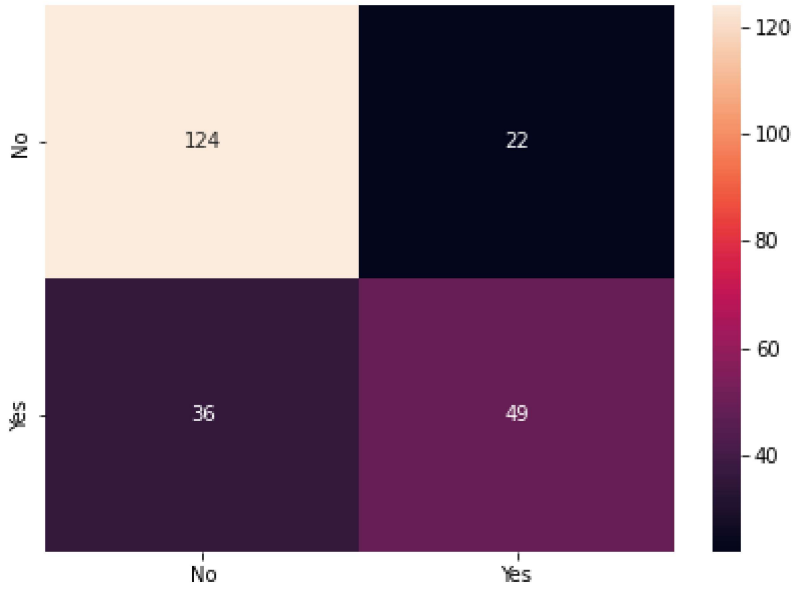
In [24]:

```
y_predict = bgcl.predict(X_test)

print(bgcl.score(X_test , y_test))

cm=metrics.confusion_matrix(y_test, y_predict,labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["No","Yes"]],
                     columns = [i for i in ["No","Yes"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
```

```
0.7489177489177489
```

Out[24]:

```
<AxesSubplot:>
```



Importing the Dependencies

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Data Collection and Analysis

PIMA Diabetes Dataset

```
# loading the diabetes dataset to a pandas DataFrame
diabetes_dataset = pd.read_csv('/content/drive/MyDrive/Diabetes_Prediction-master/Diabetes_Prediction-master/Dataset/diabetes.csv')
```

```
pd.read_csv?
```

```
# printing the first 5 rows of the dataset
diabetes_dataset.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# number of rows and Columns in this dataset
diabetes_dataset.shape
```

```
(768, 9)
```

```
# getting the statistical measures of the data
diabetes_dataset.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	7(
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	

```
diabetes_dataset['Outcome'].value_counts()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

0 --> Non-Diabetic

1 --> Diabetic

```
diabetes_dataset.groupby('Outcome').mean()
```

```
Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin       BMI  DiabetesPedigreeFunction       Age
```

```
# separating the data and labels
X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
Y = diabetes_dataset['Outcome']
```

```
print(X)
```

```
     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
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

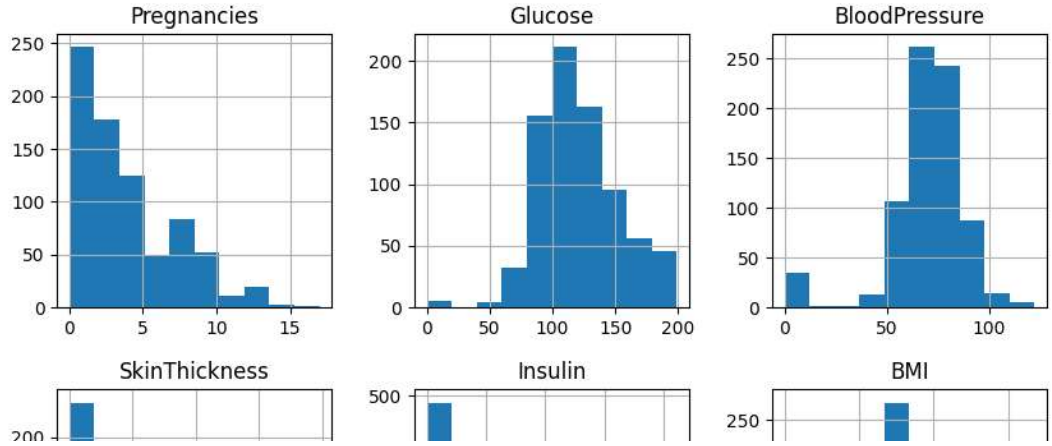
```
print(Y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

```
diabetes_dataset.hist(bins=10,figsize=(10,10))
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-44-5aa65ac69e93> in <cell line: 2>()
      1 diabetes_dataset.hist(bins=10,figsize=(10,10))
----> 2 plt.show()

NameError: name 'plt' is not defined
```

SEARCH STACK OVERFLOW



Data Standardization

```
scaler = StandardScaler()
```

```
scaler.fit(X)
```

```
▾ StandardScaler
StandardScaler()
```

```
standardized_data = scaler.transform(X)
```

```
print(standardized_data)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

```
X = standardized_data
Y = diabetes_dataset['Outcome']
```

```
print(X)
print(Y)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
0      1
1      0
2      1
3      0
```

```
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Train Test Split

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, stratify=Y, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

Training the Model

```
classifier = svm.SVC(kernel='linear')
```

```
#training the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

```
SVC
SVC(kernel='linear')
```

Model Evaluation

Accuracy Score

```
# accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
print('Accuracy score of the training data : ', training_data_accuracy)
```

```
Accuracy score of the training data :  0.7866449511400652
```

```
# accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the test data :  0.7727272727272727
```

Making a Predictive System

```
input_data = (5,166,72,19,175,25.8,0.587,51)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

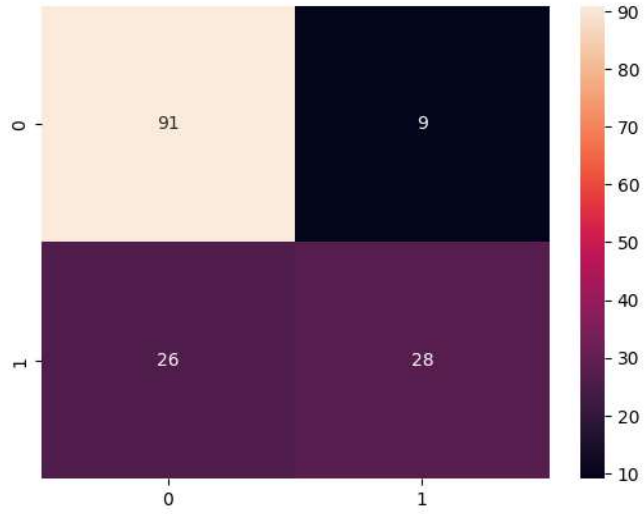
```
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person is diabetic
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted w
  warnings.warn(
```

```
# Making the Confusion Matrix
from sklearn.metrics import classification_report, confusion_matrix
cm = confusion_matrix(Y_test, X_test_prediction )

print('TP - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))))
```

```
TP - True Negative 91
FP - False Positive 9
FN - False Negative 26
TP - True Positive 28
Accuracy Rate: 0.7727272727272727
Misclassification Rate: 0.22727272727272727
```

```
import seaborn as sns
sns.heatmap(confusion_matrix(Y_test,X_test_prediction ),annot=True,fmt="d")
```

```
<Axes: >
```




```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [40]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
from sklearn.neighbors import KNeighborsClassifier
```

In [41]:
```
dataset = pd.read_csv('/content/drive/MyDrive/Diabetes_Prediction-master/Diabete
```

In [42]:
```
dataset.head()
```

Out[42]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

In [43]:
```
dataset.shape
```

Out[43]:
```
(768, 9)
```

In [44]:
```
dataset.dtypes
```

Out[44]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [45]:
```
dataset.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 [46]:
```
dataset.describe()
```

Out[46]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesP
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
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 [47]:
```
#Data Cleaning
dataset.shape
```

Out[47]: (768, 9)

In [48]:
```
dataset=dataset.drop_duplicates()
```

In [49]:
```
dataset.shape
```

Out[49]: (768, 9)

In [50]:
```
dataset.isnull().sum()
```

Out[50]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [51]:
```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 8].values
```

In [52]:
```
print('No of zero values in Glucose',dataset[dataset['Glucose']==0].shape[0])
print('No of zero values in BloodPressure',dataset[dataset['BloodPressure']==0].
print('No of zero values in SkinThickness',dataset[dataset['SkinThickness']==0].
print('No of zero values in Insulin',dataset[dataset['Insulin']==0].shape[0])
print('No of zero values in BMI',dataset[dataset['BMI']==0].shape[0])
```

```
No of zero values in Glucose 5
No of zero values in BloodPressure 35
No of zero values in SkinThickness 227
No of zero values in Insulin 374
No of zero values in BMI 11
```

In [53]:
```
df=dataset
```

In [54]:
```
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
print('No of zero values in Glucose',df[df['Glucose']==0].shape[0])
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())
print('No of zero values in BloodPressure',df[df['BloodPressure']==0].shape[0])
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].mean())
print('No of zero values in SkinThickness',df[df['SkinThickness']==0].shape[0])
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].mean())
print('No of zero values in Insulin',df[df['Insulin']==0].shape[0])
df['BMI']=df['BMI'].replace(0,df['BMI'].mean())
print('No of zero values in BMI',df[df['BMI']==0].shape[0])
```

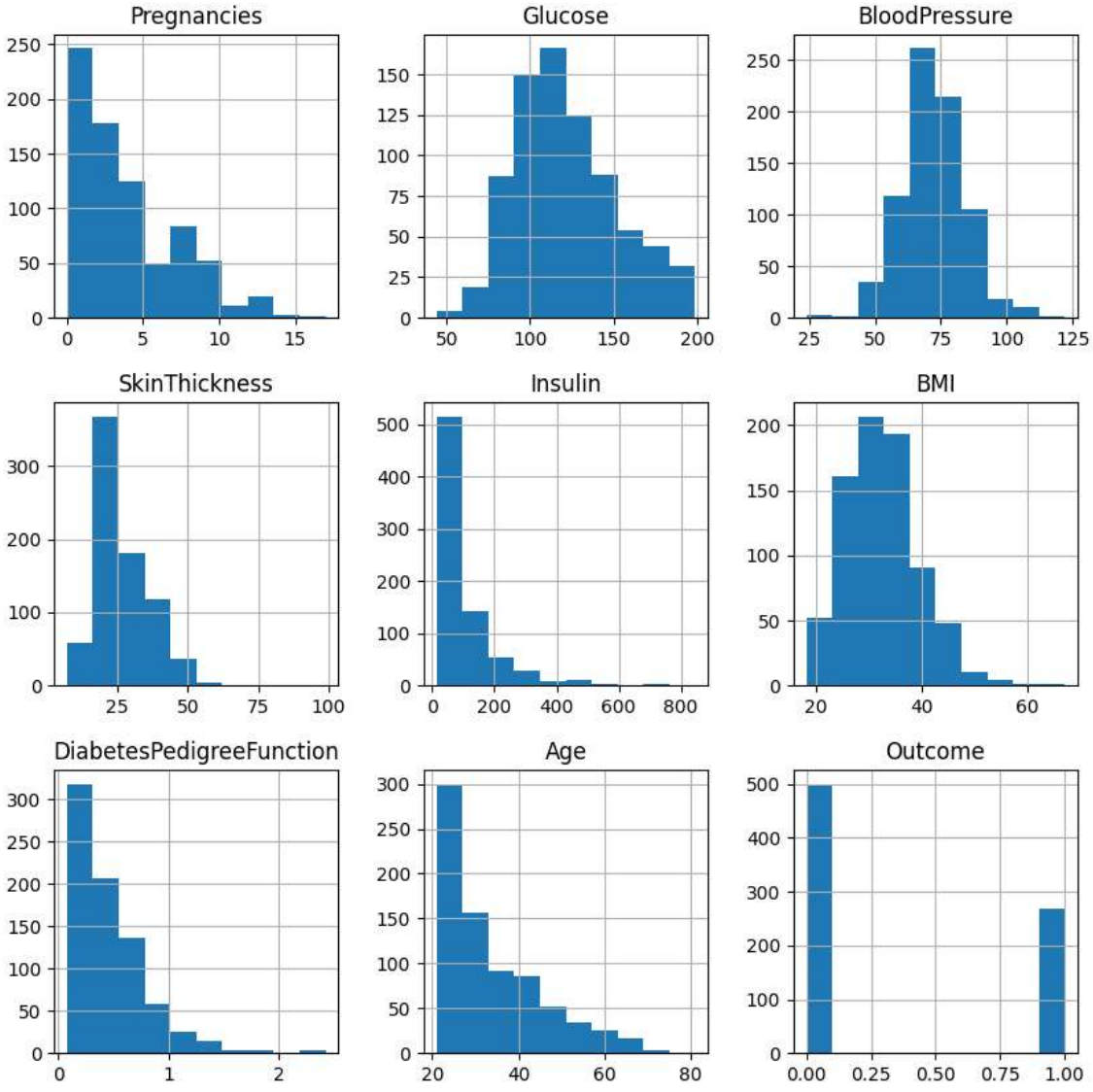
```
No of zero values in Glucose 0
No of zero values in BloodPressure 0
No of zero values in SkinThickness 0
No of zero values in Insulin 0
No of zero values in BMI 0
```

In [55]:
```
df.describe()
```

Out[55]:

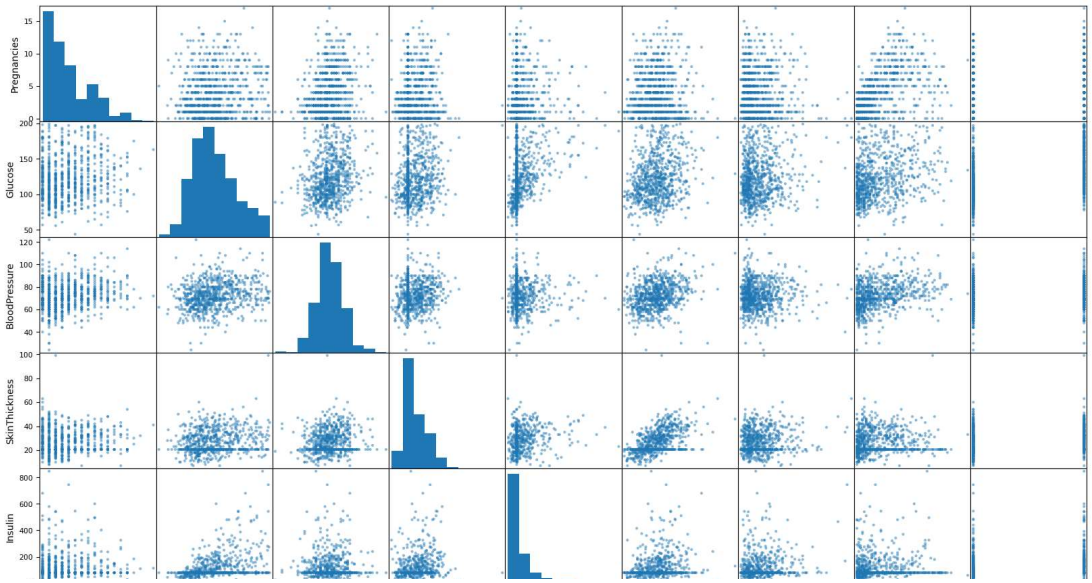
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesP
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	
50%	3.000000	117.000000	72.000000	23.000000	79.799479	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 [62]:
```
df.hist(bins=10,figsize=(10,10))
plt.show()
```



In [60]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(df,figsize=(20,20))
```

```
<Axes: xlabel='Outcome', ylabel='Outcome'>]], dtype=object)
```



In [63]:
```
sns.pairplot(data=df,hue='Outcome')
plt.show()
```

In [64]:
```
#Apply Standard Scalar
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X)
SSX=scaler.transform(X)
```

In [65]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(SSX, y, test_size = 0.2, ran
```

In [66]:
```
X_train.shape,y_train.shape
```

Out[66]: ((614, 8), (614,))

In [68]:
```
X_test.shape,y_test.shape
```

Out[68]: ((154, 8), (154,))

In [69]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
```

Out[69]:
```
KNeighborsClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [74]:
```
#Making prediction on test dataset
knn_pred=knn.predict(X_test)
knn_pred.shape
```

Out[74]:
```
(154,)
```

In [75]:
```
print("Train Accuracy of KNN",knn.score(X_train,y_train)*100)
print("Accuracy (Test) score of KNN",knn.score(X_test,y_test)*100)
print("Accuracy score of KNN",accuracy_score(y_test,knn_pred)*100)
```

```
Train Accuracy of KNN 83.87622149837134
Accuracy (Test) score of KNN 72.07792207792207
Accuracy score of KNN 72.07792207792207
```

In [77]:
```
# Predicting the Test set results
y_pred = knnClassifier.predict(X_test)
```

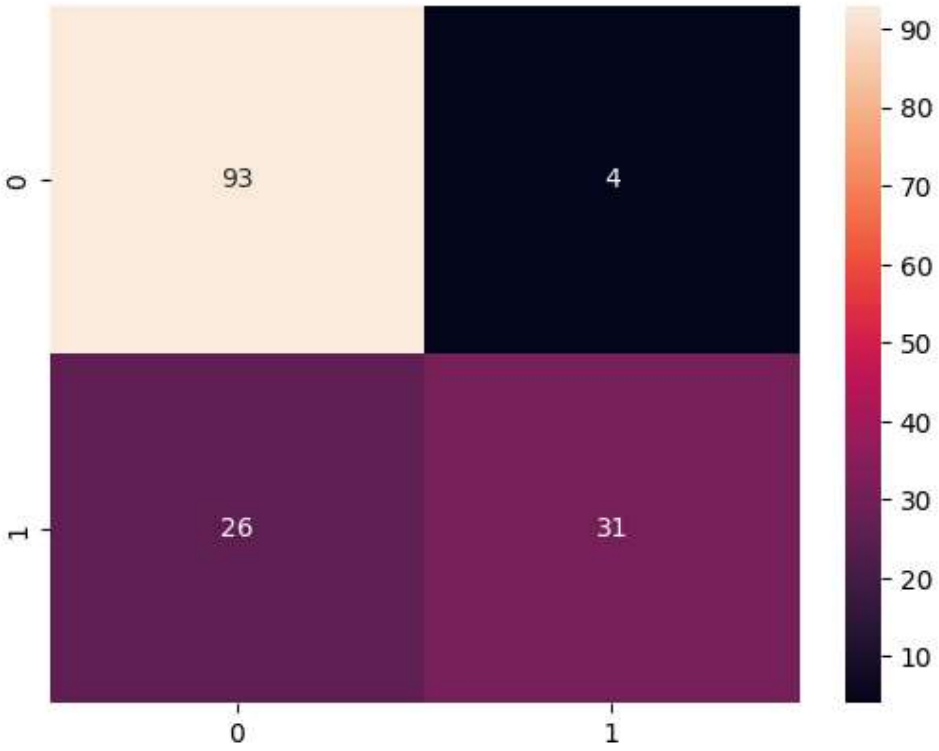
In [78]:
```
# Making the Confusion Matrix
from sklearn.metrics import classification_report, confusion_matrix
cm = confusion_matrix(y_test, y_pred)

print('TP - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm)
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np

round(roc_auc_score(y_test,y_pred),5)
```

```
TP - True Negative 93
FP - False Positive 4
FN - False Negative 26
TP - True Positive 31
Accuracy Rate: 0.8051948051948052
Misclassification Rate: 0.19480519480519481
```

Out[78]:
```
0.75131
```

In [79]:
```
sns.heatmap(confusion_matrix(y_test,y_pred),annot=True,fmt="d")
```

Out[79]: <Axes: >



In [ ]:
```

```