Description:

The objective of this diabetes dataset is to predict whether patient has diabetes or not. The datasets consist of several medical predictor(independent) variables and one target variable,(Outcome). Predictor variables includes pregancies,Gulcose,Blood Pressure,Skin Thickness, Insulin,BMI,DiabetesPedigreeFunction,age, and outcome

In [2]:

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

Load the Dataset

In [3]:

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

In [4]:

```
df
```

Out[4]:

	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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Exploratory Data Analysis(EDA): It is also known as Data Exploration is a step in the Data Analysis Process, Where a number of techniques are used to better understand the dataset being used

In this we will perform the below operations: a) Understanding Your variables b) Data Cleaning: 1) Check the duplicates 2) Check the null values

In [5]:

```
df.head()
```

Out[5]:

	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

In [6]:

```
df.tail()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

In [7]:

```
df.sample(10)
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
182	1	0	74	20	23	27.7	0.299	21	0
113	4	76	62	0	0	34.0	0.391	25	0
287	1	119	86	39	220	45.6	0.808	29	1
199	4	148	60	27	318	30.9	0.150	29	1
140	3	128	78	0	0	21.1	0.268	55	0
733	2	106	56	27	165	29.0	0.426	22	0
162	0	114	80	34	285	44.2	0.167	27	0
754	8	154	78	32	0	32.4	0.443	45	1
482	4	85	58	22	49	27.8	0.306	28	0
286	5	155	84	44	545	38.7	0.619	34	0

In [8]:

```
df.shape
```

Out[8]:

```
(768, 9)
```

In [9]:

```
df.dtypes
```

Out[9]:

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

In [10]:

```
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
```

Summary of the dataset

In [11]:

```
df.describe()
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	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	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Data Cleaning:

a) Drop the Duplicates

In [12]:

```
df.shape
```

Out[12]:

```
(768, 9)
```

In [13]:

```
df=df.drop_duplicates()
```

In [14]:

```
df.shape
```

Out[14]:

```
(768, 9)
```

Check the Null Values

In [15]:

```
df.isnull().sum()      #There is no Null Values in the given dataset
```

Out[15]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [16]:

```
df.columns
```

Out[16]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Check the number of zero values in dataset

In [17]:

```
print('No. of zero values in Glucose',df[df['Glucose']==0].shape[0])
```

```
No. of zero values in Glucose 5
```

In [18]:

```
print('No. of zero values in BloodPressure',df[df['BloodPressure']==0].shape[0])
```

```
No. of zero values in BloodPressure 35
```

In [19]:

```
print('No. of zero values in SkinThickness',df[df['SkinThickness']==0].shape[0])
```

No. of zero values in SkinThickness 227

In [20]:

```
print('No. of zero values in Insulin',df[df['Insulin']==0].shape[0])
```

No. of zero values in Insulin 374

In [21]:

```
print('No. of zero values in BMI',df[df['BMI']==0].shape[0])
```

No. of zero values in BMI 11

Replace no. of zero values with mean of that columns

In [22]:

```
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())
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].mean())
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].mean())
df['BMI']=df['BMI'].replace(0,df['BMI'].mean())
```

In [23]:

```
df.describe()
```

Out[23]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	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	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Data Visualization:

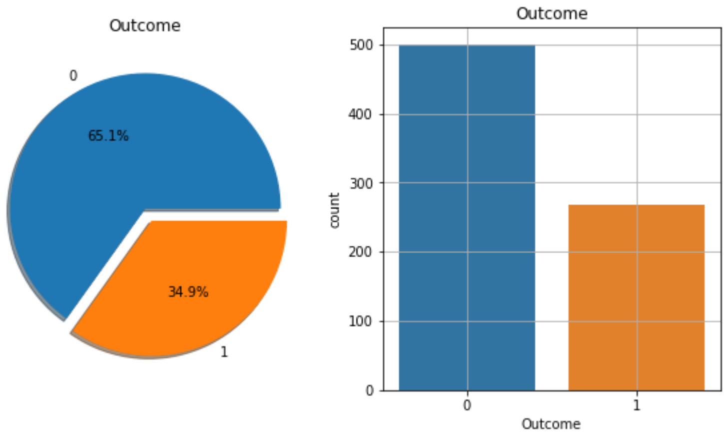
a) Count Plot

In [24]:

```
#outcome count plot    #technique used for convert imbalanced dataset to balanced dataset: undersampling, etc
#piechart
f,ax=plt.subplots(1,2,figsize=(10,5))
df['Outcome'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Outcome')
ax[0].set_ylabel('')

#countplot
sns.countplot('Outcome',data=df,ax=ax[1])
ax[1].set_title('Outcome')
N,P=df['Outcome'].value_counts()
print('Negative (0): ',N)
print('Positive (1): ',P)
plt.grid()
plt.show()
```

```
Negative (0):  500
Positive (1):  268
```



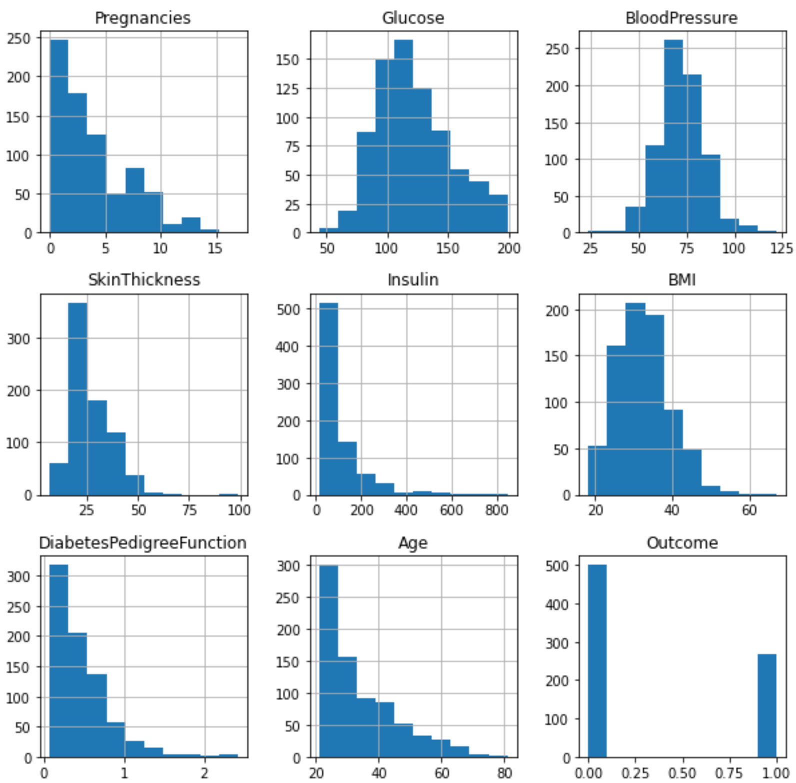
In the Outcome columns, 1 represents diabetes Positive and 0 represents diabetes negative. So,268 are diabetic (positive) and 500 are non-diabetic(negative)

the countplot tells us that the dataset is imbalanced, as number of patients who dont have diabetes is more than the those who have diabetes

Histogram

In [25]:

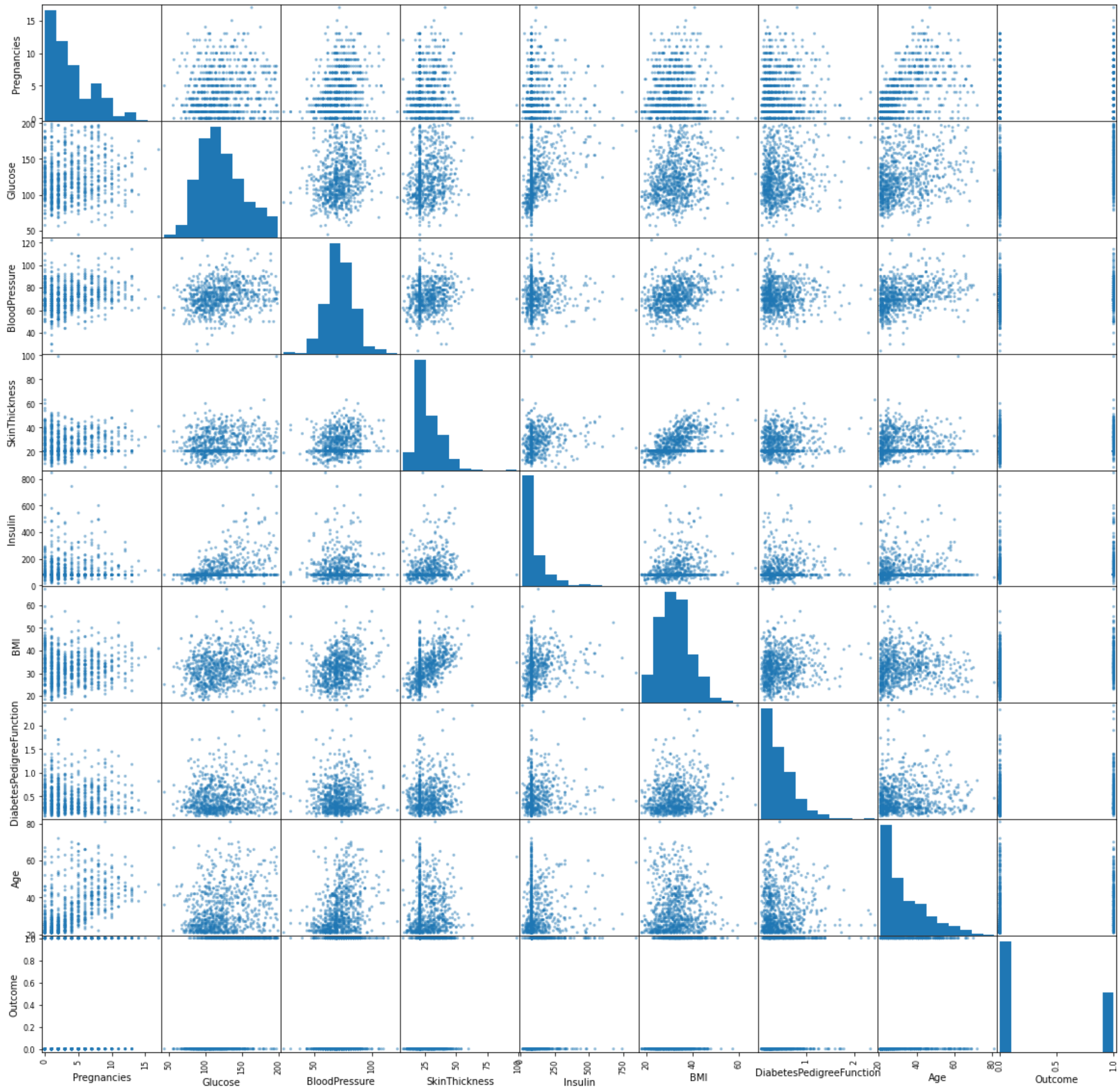
```
df.hist(bins=10,figsize=(10,10))
plt.grid()
plt.show()
```



Scatter Plot

In [26]:

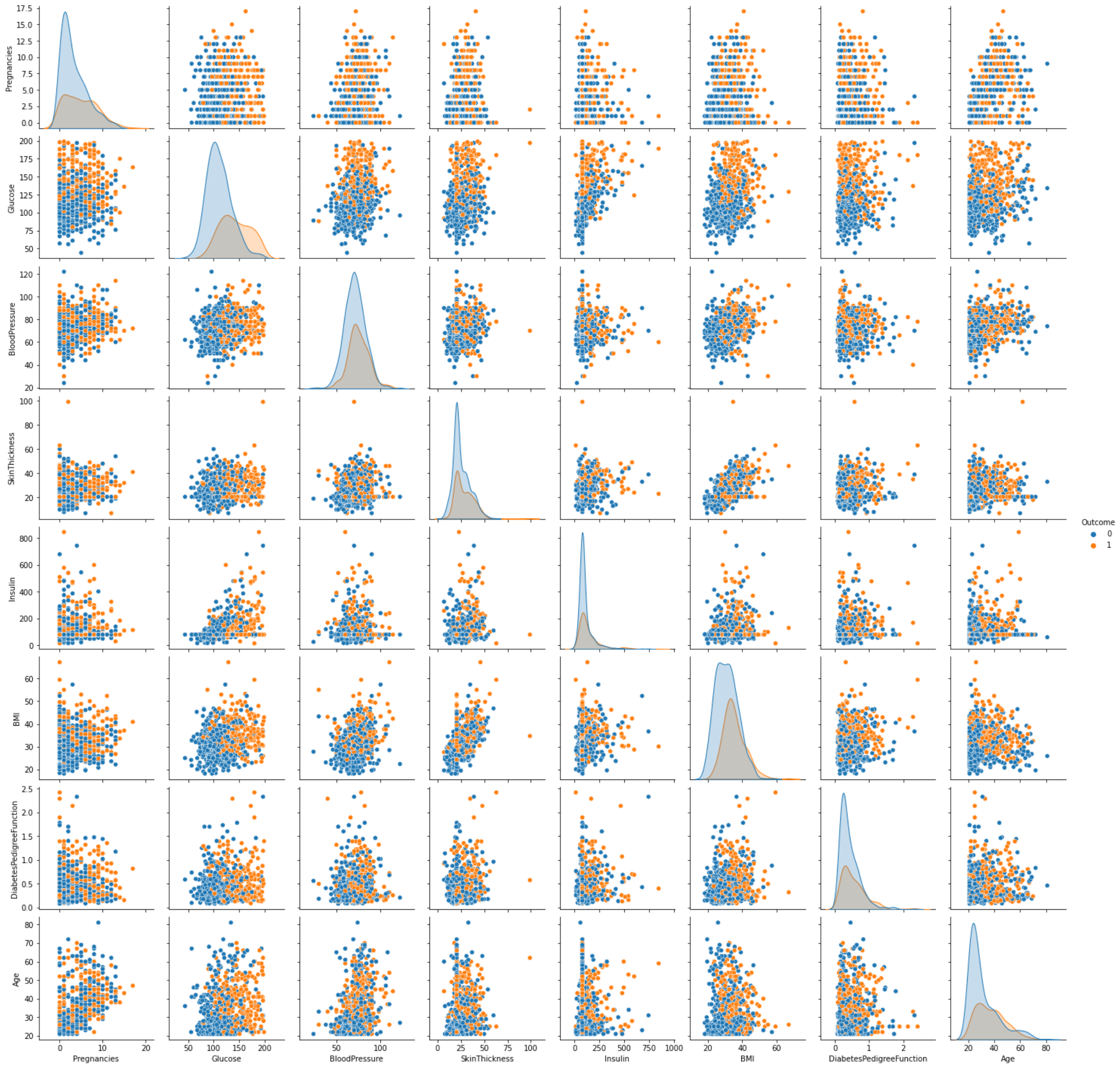
```
#scatter plot matrix
from pandas.plotting import scatter_matrix
scatter_matrix(df,figsize=(20,20));
```



Pairplot

In [27]:

```
sns.pairplot(data=df,hue='Outcome')
plt.show()
```

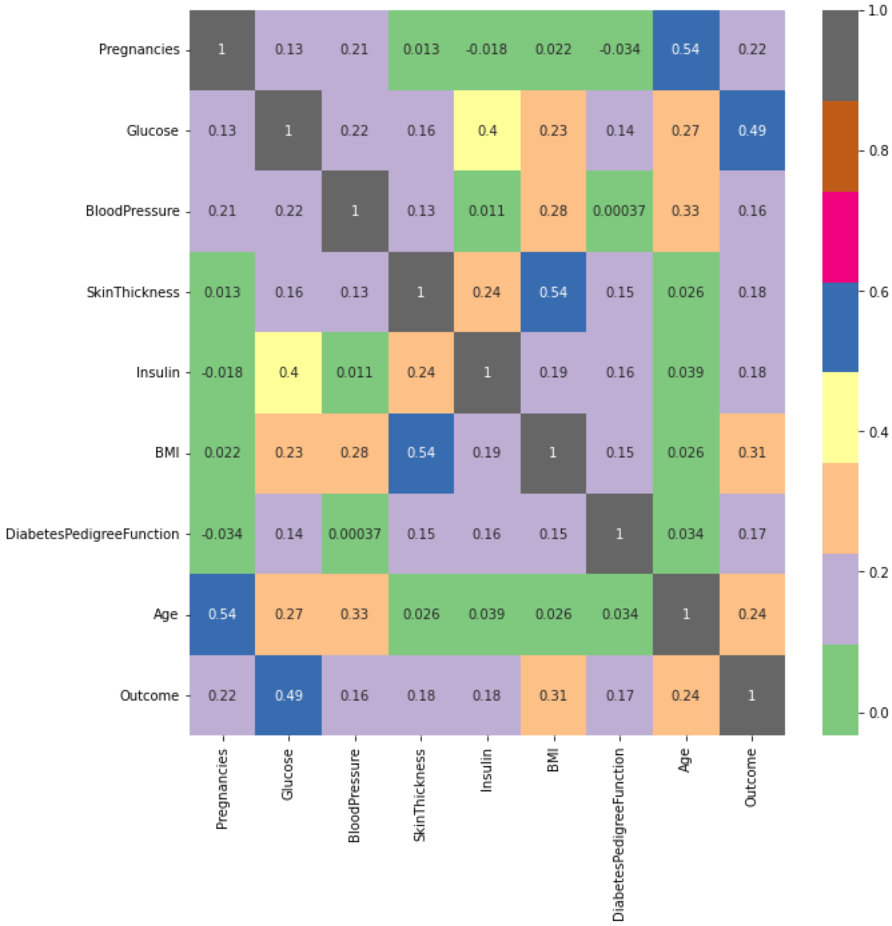


Analyzing relationships between Variables:

Correlation analysis: It is used to quantify the degree to which two variables are related. Through the correlation analysis, you evaluate correlation coefficient that tells you how much one variable changes when the other one does. Correlation analysis provides you with a linear relationship between two variables. when we correlate feature variables with the target variable, we get to know that how much dependency is there between particular feature variables and target variable

In [28]:

```
corrmat=df.corr()
top_corr_features=corrmat.index
plt.figure(figsize=(10,10))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(), annot=True, cmap="Accent")
```



Observation: from the correlation heatmap we can see that there is high correlation between outcome and [pregnancies,Glucose,BMI,Age,Insulin]. we can select these features to accept input from the user and predict the outcome

Split the data frame into X & Y

In [29]:

```
target_name='Outcome'
#separate object for target feature
y=df[target_name]

#separate object for input features
X=df.drop(target_name,axis=1)
```

In [30]:

```
X.head()
```

Out[30]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

In [31]:

```
y.head()
```

Out[31]:

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Apply Feature Scalling technique: there are 4 types of scalling normalizer,minmax scaler, binarizer and Standar Scaler

In [32]:

```
#apply standard scaler
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X)
SSX=scaler.transform(X)
```

Train Test Split

In [35]:

```
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,random_state=7)
```

In [36]:

```
X_train.shape,y_train.shape
```

Out[36]:

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

In [37]:

```
X_test.shape,y_test.shape
```

Out[37]:

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

Classification Algorithms:

Logistic Regression

In [38]:

```
from sklearn.linear_model import LogisticRegression
lr= LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train,y_train)
```

Out[38]:

```
▾ LogisticRegression
LogisticRegression(multi_class='ovr', solver='liblinear')
```

KNN Classifier

In [40]:

```
from sklearn.neighbors import KNeighborsClassifier
Knn=KNeighborsClassifier()
Knn.fit(X_train,y_train)
```

Out[40]:

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Naive_Bayes Classifier

In [41]:

```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(X_train,y_train)
```

Out[41]:

```
▾ GaussianNB
GaussianNB()
```

Support Vector Machine(SVM)

In [42]:

```
from sklearn.svm import SVC
sv=SVC()
sv.fit(X_train,y_train)
```

Out[42]:

```
▾ SVC
SVC()
```

Decision Tree

In [43]:

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

Out[43]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Random Forest

In [44]:

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(criterion='entropy')
rf.fit(X_train,y_train)
```

Out[44]:

```
▾          RandomForestClassifier
RandomForestClassifier(criterion='entropy')
```

Making Prediction:

In [45]:

```
X_test.shape
```

Out[45]:

```
(154, 8)
```

Making Prediction On Test by Using:

Logistic Regression

In [46]:

```
lr_pred=lr.predict(X_test)
```

In [47]:

```
lr_pred.shape
```

Out[47]:

```
(154,)
```

KNN

In [48]:

```
knn_pred=Knn.predict(X_test)
```

In [49]:

```
knn_pred.shape
```

Out[49]:

```
(154,)
```

Naive Bayes

In [50]:

```
nb_pred=nb.predict(X_test)
```

In [51]:

```
nb_pred.shape
```

Out[51]:

```
(154,)
```

Support Vector Machine(SVM)

In [52]:

```
sv_pred=sv.predict(X_test)
```

Decision Tree

In [53]:

```
dt_pred=dt.predict(X_test)
```

Random Forest

In [54]:

```
rf_pred=rf.predict(X_test)     #prediction of all the classifier is same (154,)
```

Model Evaluation

Train Score and Test Score

Logistic Regression

In [55]:

```
from sklearn.metrics import accuracy_score
print("Train accuracy of Logistic Regression is",lr.score(X_train,y_train)*100)
print("Test accuracy of Logistic Regression is",lr.score(X_test,y_test)*100)
```

```
Train accuracy of Logistic Regression is 77.36156351791531
Test accuracy of Logistic Regression is 77.27272727272727
```

KNN

In [56]:

```
print("Train accuracy of KNN is",Knn.score(X_train,y_train)*100)
print("Test accuracy of KNN is",Knn.score(X_test,y_test)*100)
```

```
Train accuracy of KNN is 81.10749185667753
Test accuracy of KNN is 74.67532467532467
```

Naive_Bayes

In [57]:

```
print("Train accuracy of Naive bayes is",nb.score(X_train,y_train)*100)
print("Test accuracy of Naive bayes is",nb.score(X_test,y_test)*100)
```

```
Train accuracy of Naive bayes is 74.2671009771987
Test accuracy of Naive bayes is 74.02597402597402
```

Support Vector Machine(SVM)

In [58]:

```
print("Train accuracy of SVM is",sv.score(X_train,y_train)*100)
print("Test accuracy of SVMis",sv.score(X_test,y_test)*100)       #This model is best
```

```
Train accuracy of SVM is 81.92182410423453
Test accuracy of SVMis 83.11688311688312
```

Decision Tree

In [59]:

```
print("Train accuracy of Decision Tree is",dt.score(X_train,y_train)*100)
print("Test accuracy of Decision Tree is",dt.score(X_test,y_test)*100)
```

```
Train accuracy of Decision Tree is 100.0
Test accuracy of Decision Tree is 79.87012987012987
```

Random Forest

In [60]:

```
print("Train accuracy of Random Forest is",rf.score(X_train,y_train)*100)
print("Test accuracy of Random Forest is",rf.score(X_test,y_test)*100)
print("Test accuracy of random forest is",accuracy_score(y_test,rf_pred)*100)
```

```
Train accuracy of Random Forest is 100.0
Test accuracy of Random Forest is 80.51948051948052
Test accuracy of random forest is 80.51948051948052
```

Confusion Matrix:

a) It is a table which is used to describe the performance of classification problem

b) It visualizes the accuracy of a classifier by comparing predicted values with actual values

c) The terms used in confusion metrics are:

True Positive(TP): The predicted result is positive while it is labeled as positive

False Positive(FP): The predicted result is positive while it is labeled as negative. it calls Type 1 Error as well

False Negative(FN): The predicted result is negative while it is labeled as positive. it calls Type 2 Error as well

True Negative(TN): The predicted result is negative while it is labeled as negative

Logistic Regression(Confusion Matrix)

In [ ]:

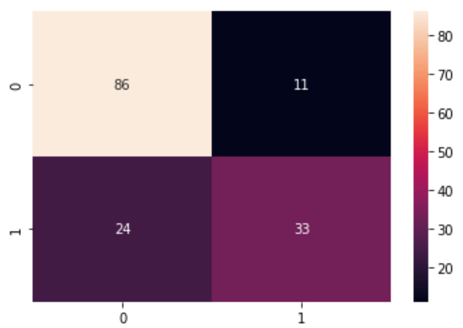
```
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,lr_pred)
cm
```

In [62]:

```
sns.heatmap(confusion_matrix(y_test,lr_pred),annot=True,fmt="d")
```

Out[62]:

<AxesSubplot:>



In [63]:

```
TN=cm[0,0]
FP=cm[0,1]
FN=cm[1,0]
TP=cm[1,1]
```

In [64]:

```
TN,FP,FN,TP
```

Out[64]:

```
(86, 11, 24, 33)
```

In [65]:

```
#making the confusion matrix of logistic regression
from sklearn.metrics import classification_report,confusion_matrix         #another method
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve

print('TN - 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))*100))
print('Error Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 86
FP - False Positive 11
FN - False Negative 24
TP - True Positive 33
Accuracy Rate: 77.27272727272727
Error Rate: 22.727272727272727
```

In [66]:

```
#Total Rate
77.27272727272727+22.727272727272727
```

Out[66]:

```
100.0
```

In [83]:

```
from sklearn.metrics import classification_report
print('Classification Report of Logistic Regression: \n', classification_report(y_test,lr_pred,digits=3))
```

```
Classification Report of Logistic Regression: 
               precision    recall  f1-score   support

           0      0.782     0.887     0.831        97
           1      0.750     0.579     0.653        57

    accuracy                          0.773       154
   macro avg      0.766     0.733     0.742       154
weighted avg      0.770     0.773     0.765       154
```

PRECISION(PPV-Positive Predictive Value): IT is the ratio of TP observations to the total (TP+FP) observations

precision=TP(TP+FP) where TP is True Positive and FP is False Positive

In [67]:

```
TP,FP
```

Out[67]:

```
(33, 11)
```

In [68]:

```
precision=TP/(TP+FP)
precision
```

Out[68]:

```
0.75
```

In [69]:

```
#print precision score

precision_score=TP/float(TP+FP)*100
print('precision score: {0:0.4f}'.format(precision_score))
```

```
precision score: 75.0000
```

In [70]:

```
from sklearn.metrics import precision_score
print("precision score is:",precision_score(y_test,lr_pred)*100)
```

```
precision score is: 75.0
```

Recall(True Positive Rate(TPR): It is ratio of correctly predicted positive (TP) observation to the total observations which are actually true

In [72]:

```
from sklearn.metrics import recall_score
print('Recall Score:',recall_score(y_test,lr_pred)*100)
```

```
Recall Score: 57.89473684210527
```

False Positive Rate(FPR)

In [73]:

```
#FPR=FP/ float(FP+TN)*100
#print('False Positive Rate: {0:0.4f}'.format(FPR))      #Trying method
```

```
False Positive Rate: 11.3402
```

In [74]:

```
FP,TN
```

Out[74]:

```
(11, 86)
```

In [76]:

```
11/(11+86)*100
```

Out[76]:

```
11.34020618556701
```

Specificity

In [77]:

```
specificity=TN/ (TN+FP)*100
print('Specificity: {0:0.4f}'.format(specificity))
```

```
Specificity: 88.6598
```

F1_Score

In [ ]:

```
from sklearn.metrics import f1_score
print('f1_score of macro: ',f1_score(y_test,lr_pred)*100)
```

In [80]:

```
print("micro average of f1_score: ", f1_score(y_test,lr_pred,average='micro')*100)
print("macro average of f1_score: ", f1_score(y_test,lr_pred,average='macro')*100)
print("weighted average score of f1_score: ", f1_score(y_test,lr_pred,average='weighted')*100)
print("non weighted average score of f1_score: ", f1_score(y_test,lr_pred,average=None)*100)
```

```
micro average of f1_score:  77.27272727272727
macro average of f1_score:  74.21916104653944
weighted average score of f1_score:  76.52373933045479
non weighted average score of f1_score:  [83.09178744 65.34653465]
```

Classification Report of Logistic Regression

In [82]:

```
from sklearn.metrics import classification_report
print('Classification Report of Logistic Regression: \n', classification_report(y_test,lr_pred,digits=3))
```

```
Classification Report of Logistic Regression:
               precision    recall  f1-score   support

           0      0.782     0.887     0.831        97
           1      0.750     0.579     0.653        57

    accuracy                          0.773       154
   macro avg      0.766     0.733     0.742       154
weighted avg      0.770     0.773     0.765       154
```

ROC Curve & ROC AUC:

ROC curve:

a) It is the important evaluating metrics that should be used to check the performance of an classification model

b) It is also called Relative operating characterstics curve because it is a comparison of two main characteristics(TPR and FPR)

c) It is plotted between sensitivity(Recall(TPR),FPR(1-Specificity)

d) ROC(Receiver Operating Characteristic) Curve tells us about how good the model can distinguish between two things

AUC:Area Under Curve (AUC) helps us to choose the best model amongst the models for which we have plotted the ROC curves

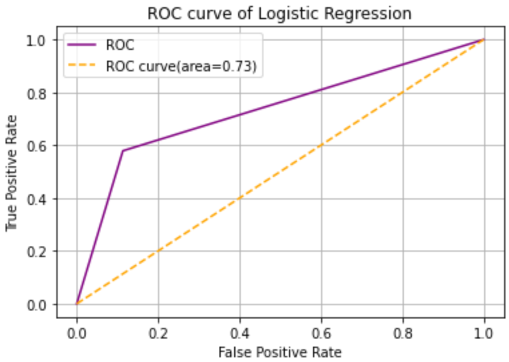
In [84]:

```
#AUC(Area Under Curve)
auc=roc_auc_score(y_test,lr_pred)
print("ROC AUC SCORE of Logistic Regression is",auc)
```

```
ROC AUC SCORE of Logistic Regression is 0.7327726532826913
```

In [85]:

```
fpr,tpr,threshold=roc_curve(y_test,lr_pred)
plt.plot(fpr,tpr,color='purple',label='ROC')
plt.plot([0,1],[0,1],color='orange', linestyle='--',label='ROC curve(area=%0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve of Logistic Regression')
plt.legend()
plt.grid()
plt.show()
```



Confusion Matrix & Classification Report of KNN

In [86]:

```
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,knn_pred)
cm
```

Out[86]:

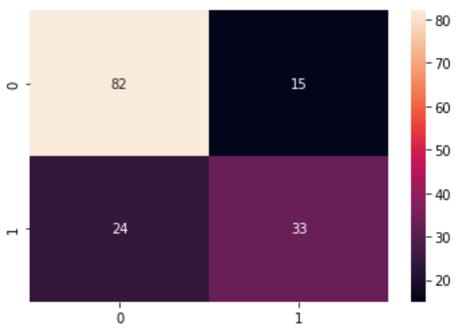
```
array([[82, 15],
       [24, 33]], dtype=int64)
```

In [89]:

```
sns.heatmap(confusion_matrix(y_test,knn_pred),annot=True,fmt="d")
```

Out[89]:

```
<AxesSubplot:>
```



In [90]:

```
TN=cm[0,0]        #2nd method
FP=cm[0,1]
FN=cm[1,0]
TP=cm[1,1]
```

In [91]:

```
TN,FP,FN,TP
```

Out[91]:

```
(82, 15, 24, 33)
```

In [94]:

```
# Accuracy & Error rate

print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Error Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))

# Classification Report
from sklearn.metrics import classification_report
print('Classification Report of KNN: \n', classification_report(y_test,knn_pred,digits=4))
```

```
Accuracy Rate: 74.67532467532467
Error Rate: 25.324675324675322
Classification Report of KNN:
               precision    recall  f1-score   support

           0     0.7736    0.8454    0.8079        97
           1     0.6875    0.5789    0.6286        57

    accuracy                         0.7468       154
   macro avg     0.7305    0.7122    0.7182       154
weighted avg     0.7417    0.7468    0.7415       154
```

In [95]:

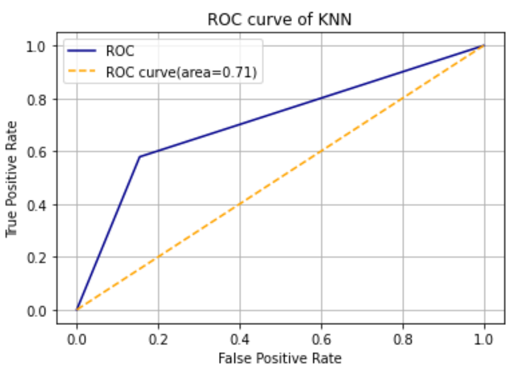
```
#AUC
auc=roc_auc_score(y_test,knn_pred)
print("ROC AUC SCORE of KNN is",auc)
```

```
ROC AUC SCORE of KNN is 0.7121540965816603
```

In [96]:

```
#ROC Curve of KNN

fpr,tpr,threshold=roc_curve(y_test,knn_pred)
plt.plot(fpr,tpr,color='darkblue',label='ROC')
plt.plot([0,1],[0,1],color='orange', linestyle='--',label='ROC curve(area=%0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve of KNN')
plt.legend()
plt.grid()
plt.show()
```



Confusion Matrix and Classification Report of Naive Bayes

In [101]:

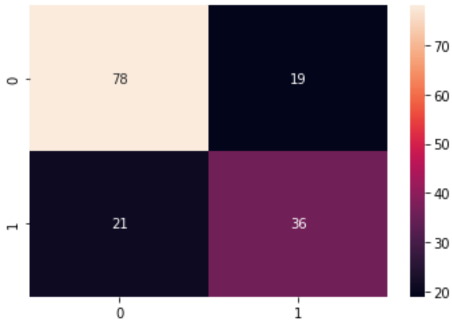
```
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,nb_pred)
#cm

#Heat Map

sns.heatmap(confusion_matrix(y_test,nb_pred),annot=True,fmt="d")
```

Out[101]:

<AxesSubplot:>



In [102]:

```
# Accuracy & Error rate

print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Error Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))

# Classification Report
from sklearn.metrics import classification_report
print('Classification Report of Naive Bayes: \n', classification_report(y_test,nb_pred,digits=4))
```

```
Accuracy Rate: 74.02597402597402
Error Rate: 25.97402597402597
Classification Report of Naive Bayes:
               precision    recall  f1-score   support

           0     0.7879    0.8041    0.7959        97
           1     0.6545    0.6316    0.6429        57

    accuracy                         0.7403       154
   macro avg     0.7212    0.7179    0.7194       154
weighted avg     0.7385    0.7403    0.7393       154
```

In [103]:

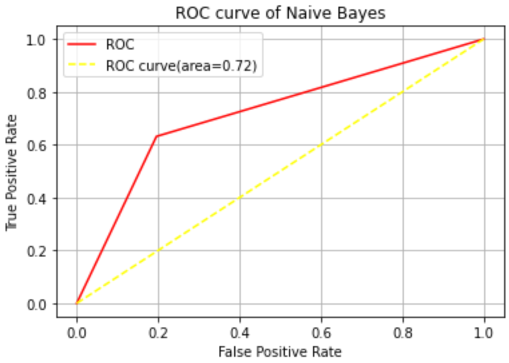
```
#AUC
auc=roc_auc_score(y_test,nb_pred)
print("ROC AUC SCORE of Naive Bayes is",auc)
```

```
ROC AUC SCORE of Naive Bayes is 0.7178513293543136
```

In [104]:

```
#ROC Curve of Naive Bayes

fpr,tpr,threshold=roc_curve(y_test,nb_pred)
plt.plot(fpr,tpr,color='Red',label='ROC')
plt.plot([0,1],[0,1],color='Yellow', linestyle='--',label='ROC curve(area=%0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve of Naive Bayes')
plt.legend()
plt.grid()
plt.show()
```



Confusion Matrix and Classification Report of SVM

In [105]:

```
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,sv_pred)
cm
```

Out[105]:

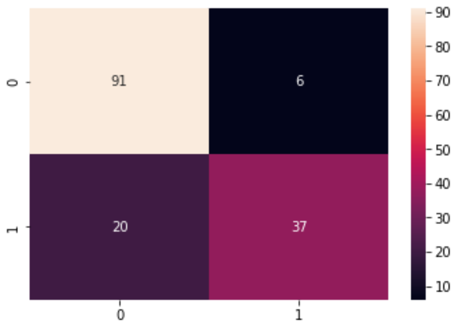
```
array([[91,  6],
       [20, 37]], dtype=int64)
```

In [106]:

```
#Heat Map
sns.heatmap(confusion_matrix(y_test,sv_pred),annot=True,fmt="d")
```

Out[106]:

```
<AxesSubplot:>
```



In [107]:

```
# Accuracy & Error rate

print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Error Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))

# Classification Report
from sklearn.metrics import classification_report
print('Classification Report of SVM: \n', classification_report(y_test,sv_pred,digits=4))
```

```
Accuracy Rate: 83.11688311688312
Error Rate: 16.883116883116884
Classification Report of SVM:
               precision    recall  f1-score   support

           0     0.8198    0.9381    0.8750        97
           1     0.8605    0.6491    0.7400        57

    accuracy                         0.8312       154
   macro avg     0.8401    0.7936    0.8075       154
weighted avg     0.8349    0.8312    0.8250       154
```

In [108]:

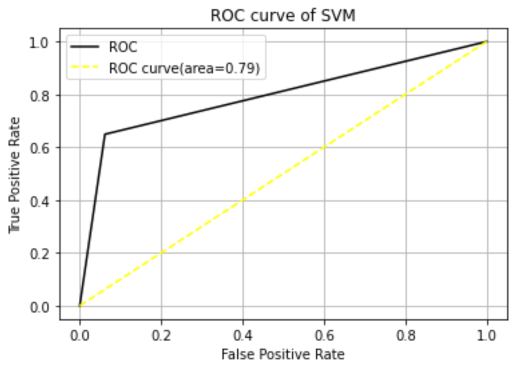
```
#AUC
auc=roc_auc_score(y_test,sv_pred)
print("ROC AUC SCORE of SVM is",auc)
```

```
ROC AUC SCORE of SVM is 0.7936335684572255
```

In [109]:

```
#ROC Curve of SVM

fpr,tpr,threshold=roc_curve(y_test,sv_pred)
plt.plot(fpr,tpr,color='Black',label='ROC')
plt.plot([0,1],[0,1],color='Yellow', linestyle='--',label='ROC curve(area=%0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve of SVM')
plt.legend()
plt.grid()
plt.show()
```



Confusion Matrix & Classification Report of Decision Tree

In [110]:

```
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,dt_pred)
cm
```

Out[110]:

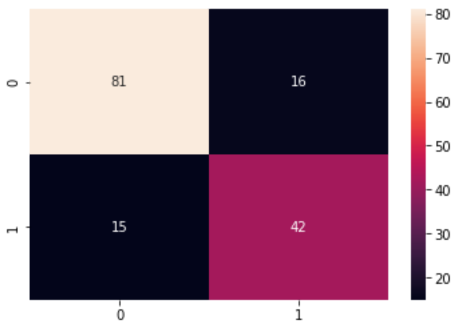
```
array([[81, 16],
       [15, 42]], dtype=int64)
```

In [112]:

```
#Heat Map
sns.heatmap(confusion_matrix(y_test,dt_pred),annot=True,fmt="d")
```

Out[112]:

```
<AxesSubplot:>
```



In [113]:

```
# Accuracy & Error rate

print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Error Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))

# Classification Report
from sklearn.metrics import classification_report
print('Classification Report of Decision Tree: \n', classification_report(y_test,dt_pred,digits=4))
```

```
Accuracy Rate: 79.87012987012987
Error Rate: 20.12987012987013
Classification Report of Decision Tree:
               precision    recall  f1-score   support

           0     0.8438    0.8351    0.8394        97
           1     0.7241    0.7368    0.7304        57

    accuracy                         0.7987       154
   macro avg     0.7839    0.7859    0.7849       154
weighted avg     0.7995    0.7987    0.7991       154
```

In [114]:

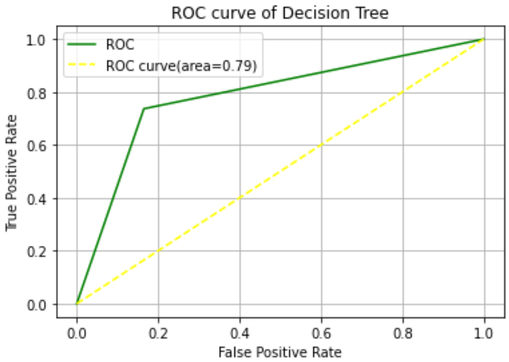
```
#AUC
auc=roc_auc_score(y_test,dt_pred)
print("ROC AUC SCORE of Decision Tree is",auc)
```

```
ROC AUC SCORE of Decision Tree is 0.7859468258274552
```

In [116]:

```
#ROC Curve of Decision Tree

fpr,tpr,threshold=roc_curve(y_test,dt_pred)
plt.plot(fpr,tpr,color='green',label='ROC')
plt.plot([0,1],[0,1],color='Yellow', linestyle='--',label='ROC curve(area=%0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve of Decision Tree')
plt.legend()
plt.grid()
plt.show()
```



Confusion Matrix and Classification Report of Random Forest

In [118]:

```
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,rf_pred)
cm
```

Out[118]:

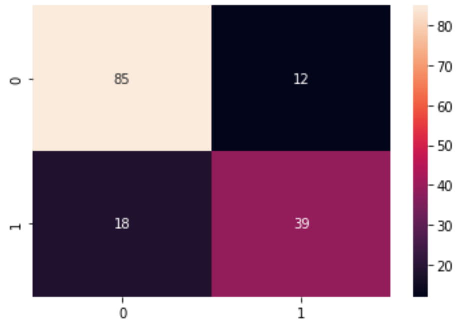
```
array([[85, 12],
       [18, 39]], dtype=int64)
```

In [122]:

```
#Heat Map
sns.heatmap(confusion_matrix(y_test,rf_pred),annot=True,fmt="d")
```

Out[122]:

```
&lt;AxesSubplot:&gt;
```



In [120]:

```
# Accuracy & Error rate

print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Error Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))

# Classification Report
from sklearn.metrics import classification_report
print('Classification Report of Decision Tree: \n', classification_report(y_test,rf_pred,digits=4))
```

```
Accuracy Rate: 80.51948051948052
Error Rate: 19.480519480519483
Classification Report of Decision Tree:
               precision    recall  f1-score   support

           0     0.8252    0.8763    0.8500        97
           1     0.7647    0.6842    0.7222        57

    accuracy                         0.8052       154
   macro avg     0.7950    0.7802    0.7861       154
weighted avg     0.8028    0.8052    0.8027       154
```

In [123]:

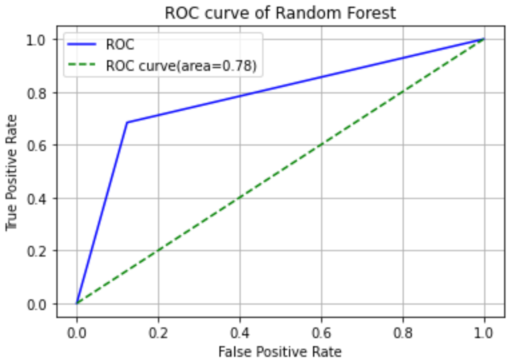
```
#AUC
auc=roc_auc_score(y_test,rf_pred)
print("ROC AUC SCORE of Random Forest is",auc)
```

```
ROC AUC SCORE of Random Forest is 0.7802495930548019
```

In [124]:

```
#ROC Curve of Random Forest

fpr,tpr,threshold=roc_curve(y_test,rf_pred)
plt.plot(fpr,tpr,color='blue',label='ROC')
plt.plot([0,1],[0,1],color='green', linestyle='--',label='ROC curve(area=%0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve of Random Forest')
plt.legend()
plt.grid()
plt.show()
```



END

In [ ]: