In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
data = pd.read_csv("C:/Users/anjal/Desktop/diabetes.csv",sep=",")
```

In [3]:
```
data.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```
data.head(5)
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
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	

In [5]:
```
data.columns ## to see all the columns present in the data
```

Out[5]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [6]:
```
data.dtypes ##to see the datatypes of each col
```

Out[6]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [7]:
```
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
```

In [8]:
```
data.describe() ##using numpy calculate all math functions
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Di
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 [9]:
```
data.isna().sum() ##to check null values and their sums
```

Out[9]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [10]:
```
data.isna().any()
```

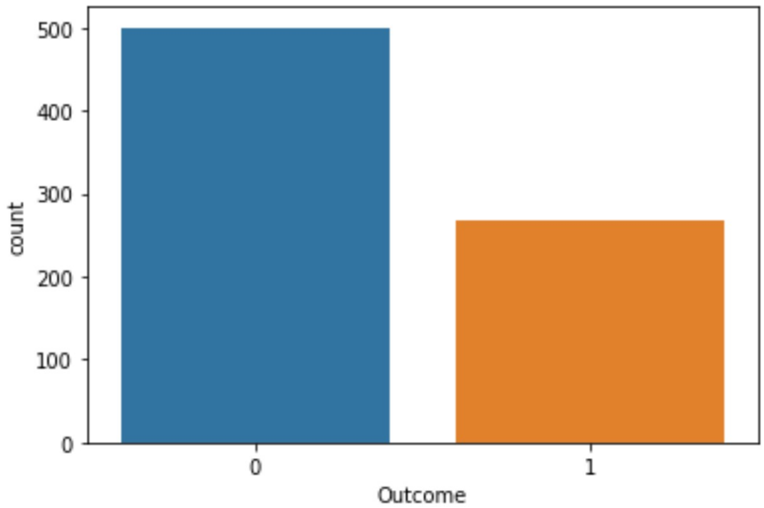
Out[10]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

Data analysis and visualization

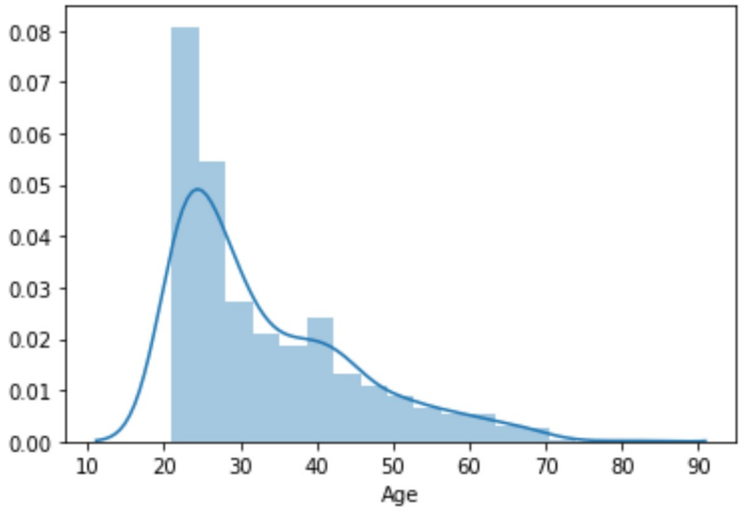
In [11]:
```
data.Outcome.value_counts() ##outcome is the classification colmn an
d value_counts count the no of discrete outcomes.
```

Out[11]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

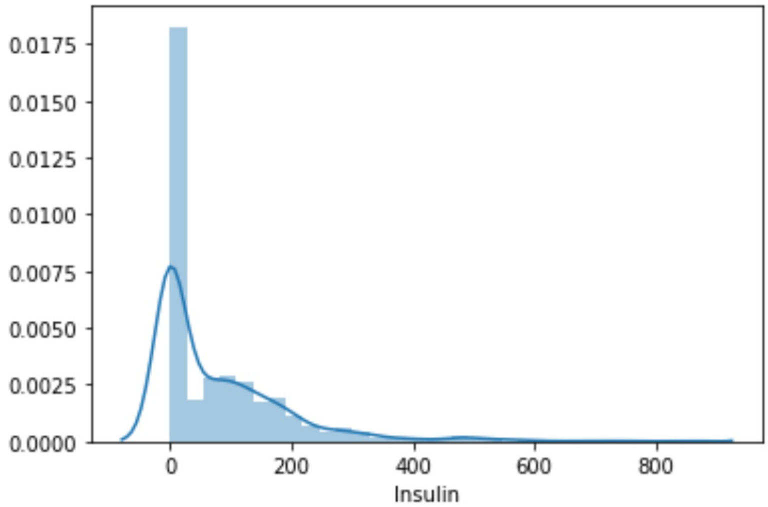
In [12]:
```
sns.countplot(data.Outcome,data=data)
plt.show() ## plot the above data using seaborn
```



In [13]:
```
sns.distplot(data['Age'],kde=True)
plt.show()  ##KDE find the density of the variable(age) using probab
ility density function and gaussian algo
```

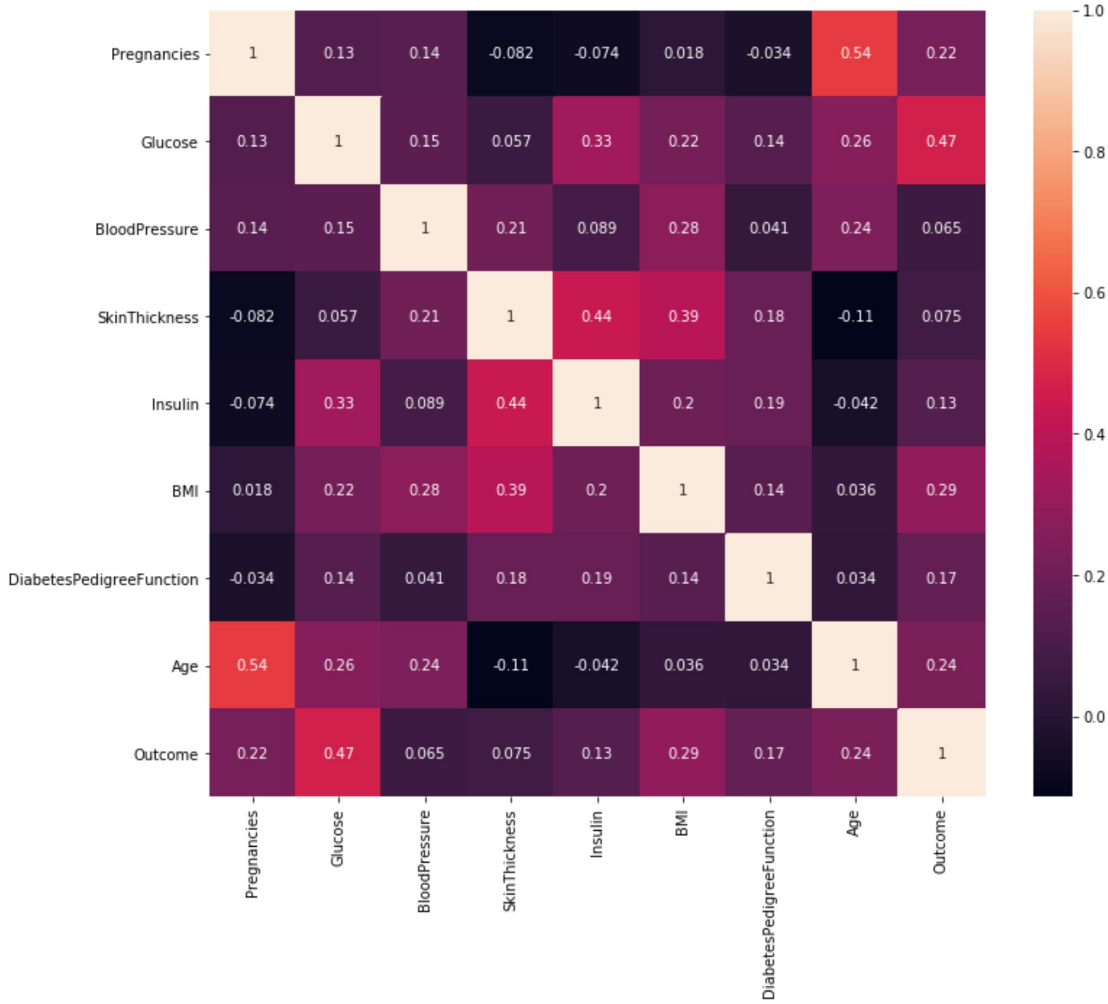


In [14]:
```
sns.distplot(data['Insulin'],kde=True)
plt.show()
```



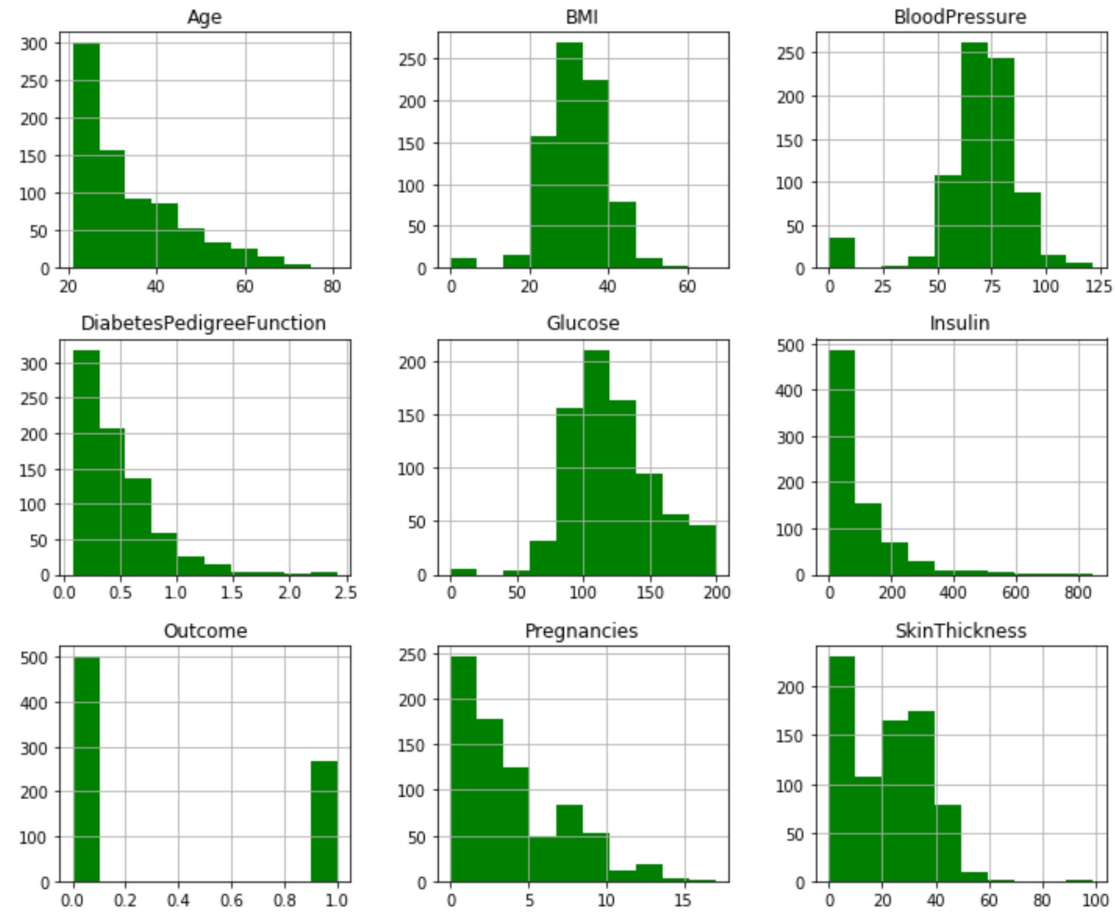
In [15]:
```
correlation = data.corr()
plt.figure(figsize = (12,10))
sns.heatmap(correlation,annot = True)  ##correlation used to find th
e dependencies of each independent variable(columns) on each other
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x205360da548>

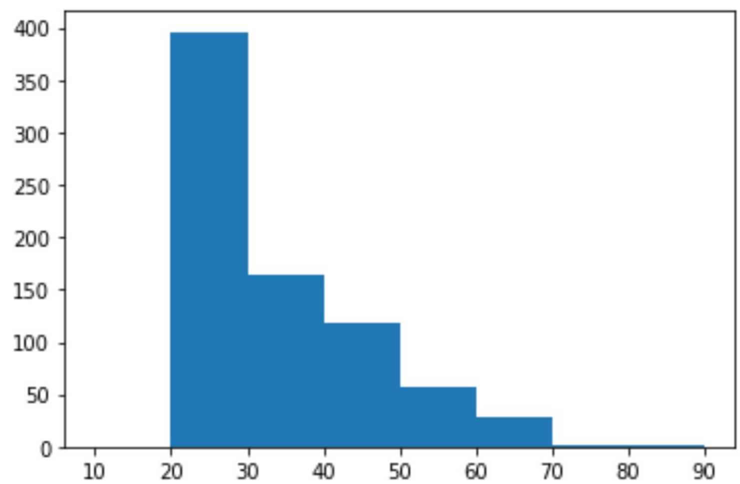


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
SkinThickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
DiabetesPedigreeFunction	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

In [16]:
```
data.hist(figsize = (12,10),color = 'green')
plt.show() ##hist plot for each column
```



In [17]:
```
plt.hist(data.Age,bins = [10,20,30,40,50,60,70,80,90])
plt.show() ## bins is the interval between which the histogram needs
to be plotted
```

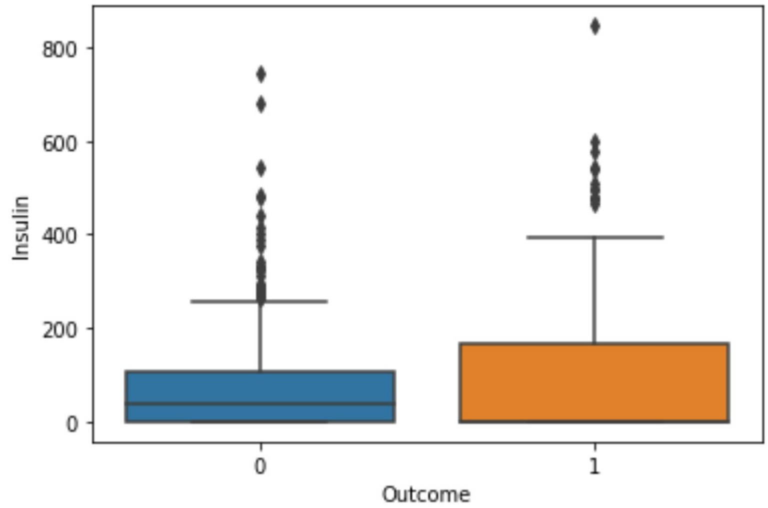


In [18]:
```
data.groupby("Outcome").mean() ##mean of each variable according to
outcome
```

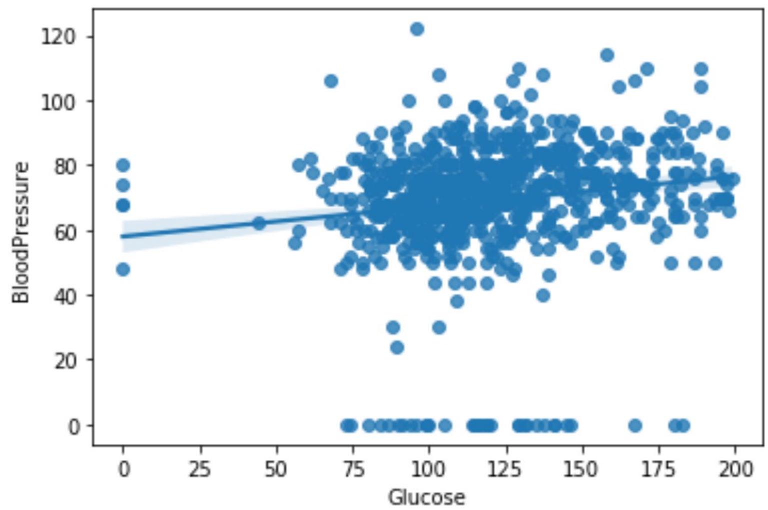
Out[18]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
Outcome						
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537

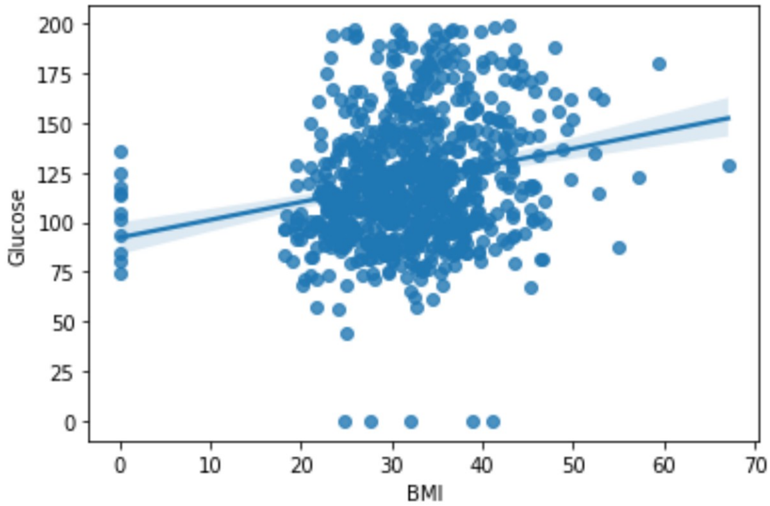
In [19]:
```
# box plot between outcome and insulin
sns.boxplot(x='Outcome',y='Insulin',data=data)
plt.show()
```



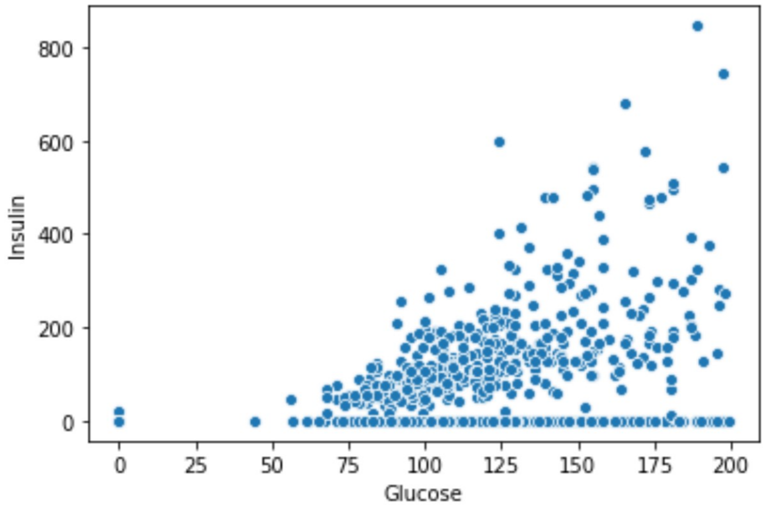
In [20]:
```
# scatter plot between glucose and  Blood pressure with regression l
ine
sns.regplot(x = "Glucose",y = "BloodPressure",data = data)
plt.show()
```



In [21]:
```
sns.regplot(x='BMI', y= 'Glucose', data=data)
plt.show()
```



In [22]:
```
sns.scatterplot(x='Glucose', y= 'Insulin', data=data) # scatter plot
plt.show()
```



In [23]:
```
sns.pairplot(data,hue='Outcome')
plt.show()
```

Now we are done with EDA and we'll start predicting using various algorithm

In [24]:
```
X = data.iloc[:,:8] ##to locate data
Y = data.iloc[:,8]
```

In [25]:
```
from sklearn.svm import SVC ##using SKlearn library to import and us
e various ML algorithms
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
```

In [26]:
```
scaler = StandardScaler()
standardized_data = scaler.fit_transform(X)  ##scaling the data usin
g PCA
```

In [27]:
```
pd.DataFrame(standardized_data,columns=['Pregnancies', 'Glucose', 'B
loodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']).head() ##Dimension
reduction is performed and dataset is scaled
```

Out[27]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	

In [28]:
```
train_x,test_x,train_y,test_y = train_test_split(standardized_data,
Y,test_size = 0.3,random_state = 32) ##splitting a dataset
```

In [29]:
```
print("Shape of train x: ",train_x.shape)
print("Shape of train y: ",train_y.shape)

print("Shape of test x: ",test_x.shape)
print("Shape of test y: ",test_y.shape)
```

```
Shape of train x:  (537, 8)
Shape of train y:  (537,)
Shape of test x:  (231, 8)
Shape of test y:  (231,)
```

SVM MODEL

In [30]:
```
# defining our model
model = SVC()
model.fit(train_x,train_y)
```

Out[30]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, co
ef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel
='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=T
rue,
    tol=0.001, verbose=False)
```

In [31]:
```
# training accuracy
X_train_predicted = model.predict(train_x)
print("Accuracy of training data is: ",accuracy_score(train_y,X_trai
n_predicted))
```

```
Accuracy of training data is:  0.8342644320297952
```

In [32]:
```
# prediction for test data
predicted = model.predict(test_x)
```

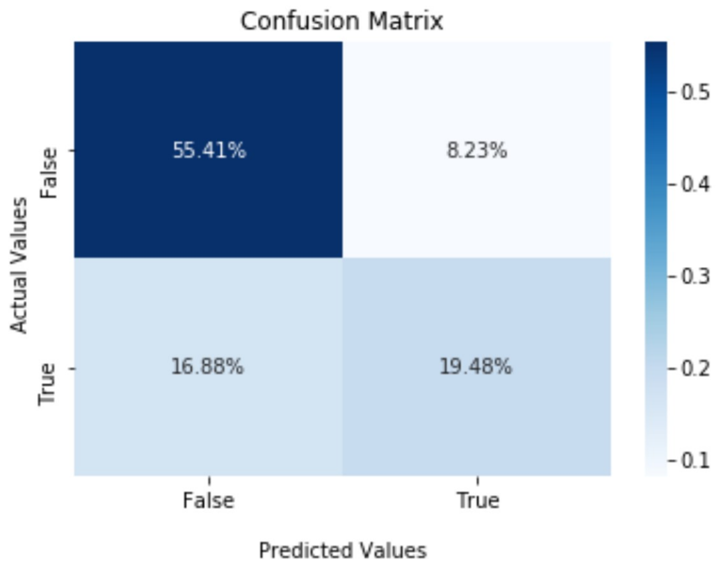
In [33]:
```
svm_test_score = accuracy_score(test_y,predicted)
svm_test_score
```

Out[33]:
```
0.7489177489177489
```

In [34]:
```
print(classification_report(test_y,predicted))
```

```
              precision    recall  f1-score   support

           0       0.77      0.87      0.82       147
           1       0.70      0.54      0.61        84

    accuracy                           0.75       231
   macro avg       0.73      0.70      0.71       231
weighted avg       0.74      0.75      0.74       231
```

In [35]:
```
import seaborn as sns
cm =  confusion_matrix(test_y,predicted)
ax = sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues')

ax.set_title('Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



LOGISTIC REGRESSION

In [36]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

In [37]:
```
lr.fit(train_x,train_y)
```

Out[37]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_inter
cept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10
0,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)
```

In [38]:
```
# training accuracy
X_train_predicted = lr.predict(train_x)
print("Accuracy of training data is: ",accuracy_score(train_y,X_trai
n_predicted))
```

```
Accuracy of training data is:  0.7802607076350093
```

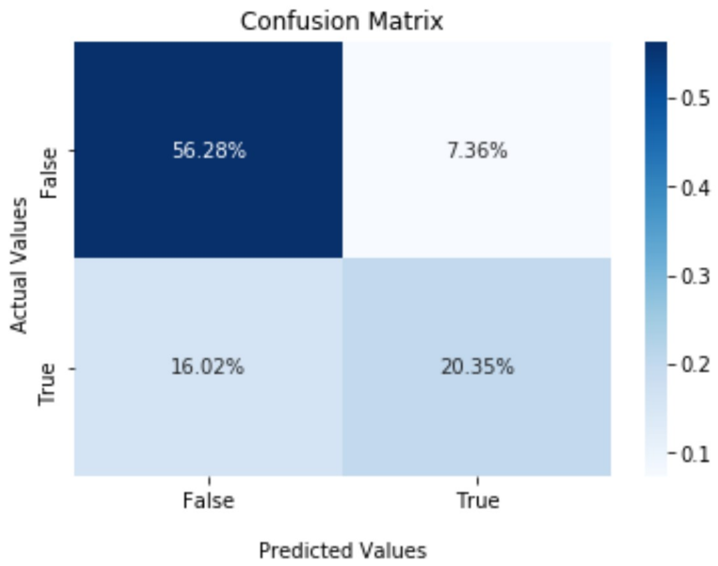
In [39]:
```
# prediction for test data
predicted = lr.predict(test_x)
lr_test_score = accuracy_score(test_y,predicted)
lr_test_score
```

Out[39]:
```
0.7662337662337663
```

In [40]:
```
print(classification_report(test_y,predicted))
```

```
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       147
           1       0.73      0.56      0.64        84

    accuracy                           0.77       231
   macro avg       0.76      0.72      0.73       231
weighted avg       0.76      0.77      0.76       231
```

In [41]:
```
import seaborn as sns
cm =  confusion_matrix(test_y,predicted)
ax = sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues')

ax.set_title('Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



RANDOM FOREST

In [42]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [43]:
```
rf = RandomForestClassifier(n_estimators = 50,max_leaf_nodes = 66,ma
x_samples = 66)
rf.fit(train_x,train_y)
```

Out[43]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight
=None,
                       criterion='gini', max_depth=None, max_featu
res='auto',
                       max_leaf_nodes=66, max_samples=66,
                       min_impurity_decrease=0.0, min_impurity_spl
it=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=
50,
                       n_jobs=None, oob_score=False, random_state=
None,
                       verbose=0, warm_start=False)
```

In [44]:
```
# training accuracy
X_train_predicted = rf.predict(train_x)
print("Accuracy of training data is: ",accuracy_score(train_y,X_trai
n_predicted))
```

```
Accuracy of training data is:  0.8212290502793296
```

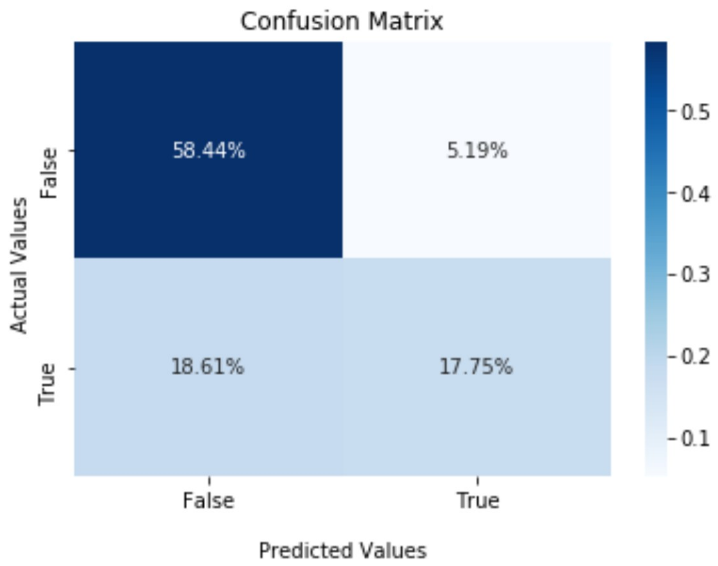
In [45]:
```
# prediction for test data
predicted = rf.predict(test_x)
rf_test_score = accuracy_score(test_y,predicted)
rf_test_score
```

Out[45]:
```
0.7619047619047619
```

In [46]:
```
print(classification_report(test_y,predicted))
```

```
              precision    recall  f1-score   support

           0       0.76      0.92      0.83       147
           1       0.77      0.49      0.60        84

    accuracy                           0.76       231
   macro avg       0.77      0.70      0.71       231
weighted avg       0.76      0.76      0.75       231
```

In [47]:
```
import seaborn as sns
cm =  confusion_matrix(test_y,predicted)
ax = sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues')

ax.set_title('Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



DECISION TREE

In [48]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [49]:
```
dt = DecisionTreeClassifier(criterion='entropy',min_samples_split =
6, min_samples_leaf = 25)
dt.fit(train_x,train_y)
```

Out[49]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion
='entropy',
                       max_depth=None, max_features=None, max_leaf
_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spl
it=None,
                       min_samples_leaf=25, min_samples_split=6,
                       min_weight_fraction_leaf=0.0, presort='depr
ecated',
                       random_state=None, splitter='best')
```

In [50]:
```
# training accuracy
X_train_predicted = dt.predict(train_x)
print("Accuracy of training data is: ",accuracy_score(train_y,X_trai
n_predicted))
```

```
Accuracy of training data is:  0.7858472998137802
```

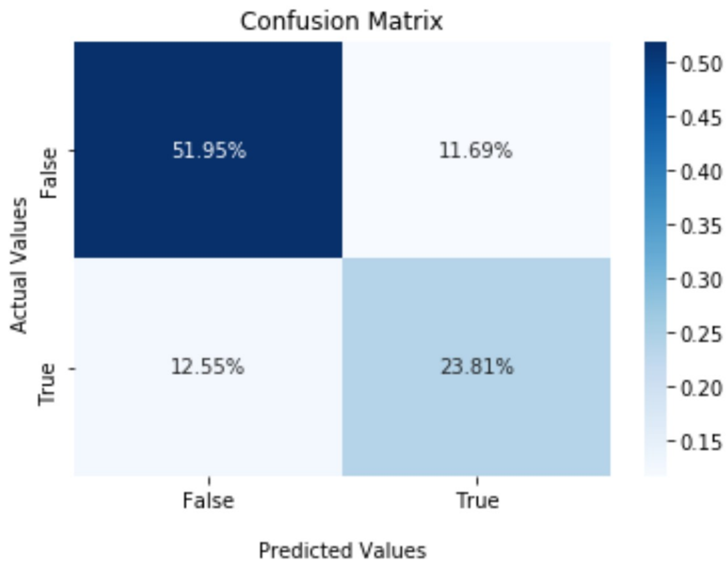
In [51]:
```
# prediction for test data
predicted = dt.predict(test_x)
dt_test_score = accuracy_score(test_y,predicted)
dt_test_score
```

Out[51]:
```
0.7575757575757576
```

In [52]:
```
print(classification_report(test_y,predicted))
```

```
              precision    recall  f1-score   support

           0       0.81      0.82      0.81       147
           1       0.67      0.65      0.66        84

    accuracy                           0.76       231
   macro avg       0.74      0.74      0.74       231
weighted avg       0.76      0.76      0.76       231
```

In [53]:
```
import seaborn as sns
cm =  confusion_matrix(test_y,predicted)
ax = sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues')

ax.set_title('Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



KNN

In [54]:
```
from sklearn.neighbors import KNeighborsClassifier

neighbors = [1,2,3,5,7,9,10,12,15,19,21]
test_scores = []
train_scores = []

for i in neighbors:

    knn = KNeighborsClassifier(i)
    knn.fit(train_x,train_y)

    train_scores.append(knn.score(train_x,train_y))
    test_scores.append(knn.score(test_x,test_y))
```

In [55]:
```
ind=np.argmax(test_scores)
```

In [56]:
```
# coressponding train and test scores
print("Test score: ",test_scores[ind])
print("Train score: ",train_scores[ind])
```

```
Test score:  0.7662337662337663
Train score:  0.7690875232774674
```

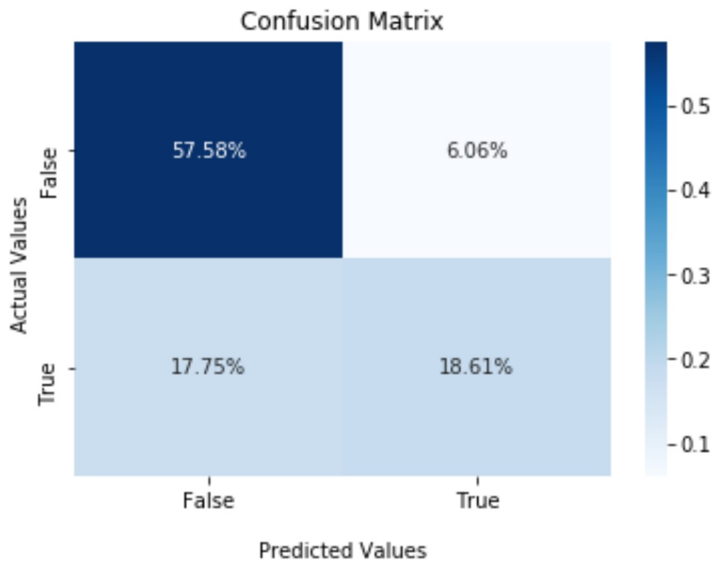
In [57]:
```
# prediction for test data
predicted = knn.predict(test_x)
knn_test_score = accuracy_score(test_y,predicted)
knn_test_score
```

Out[57]: 0.7619047619047619

In [58]:
```
print(classification_report(test_y,predicted))
```

```
              precision    recall  f1-score   support

           0       0.76      0.90      0.83       147
           1       0.75      0.51      0.61        84

    accuracy                           0.76       231
   macro avg       0.76      0.71      0.72       231
weighted avg       0.76      0.76      0.75       231
```

In [59]:
```
import seaborn as sns
cm =  confusion_matrix(test_y,predicted)
ax = sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues')

ax.set_title('Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



MODELS COMPARING

In [60]:
```
# comparing our all models
models = {"SVM":svm_test_score, "Logistic Regression":lr_test_scor
e,"Random Forest":rf_test_score,"Decision Tree": dt_test_score,
         "KNN":knn_test_score}
model = pd.DataFrame({"Models":['SVM',"Logistic Regression","Random
Forest","Decision Tree","KNN"],
                     "Score":[svm_test_score,lr_test_score,rf_test_s
core,dt_test_score,knn_test_score]},index = np.arange(1,6))
model.head(5)
```

Out[60]:

	Models	Score
1	SVM	0.748918
2	Logistic Regression	0.766234
3	Random Forest	0.761905
4	Decision Tree	0.757576
5	KNN	0.761905

In [61]:

```
model.sort_values(by='Score',ascending = False)
## Random Forest is our best model
```

Out[61]:

	Models	Score
2	Logistic Regression	0.766234
3	Random Forest	0.761905
5	KNN	0.761905
4	Decision Tree	0.757576
1	SVM	0.748918

In [ ]: