In [9]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

In [10]:
```
df=pd.read_csv("health care diabetes.csv")
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [12]:
```
df.head()
```

Out[12]:

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

In [13]:
```
df.dtypes
```

Out[13]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [14]:
```
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
```

In [15]:
```
df.corr()
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695

1. Perform descriptive analysis. Understand the variables and their corresponding values.

In [16]:
```
df.describe()
```

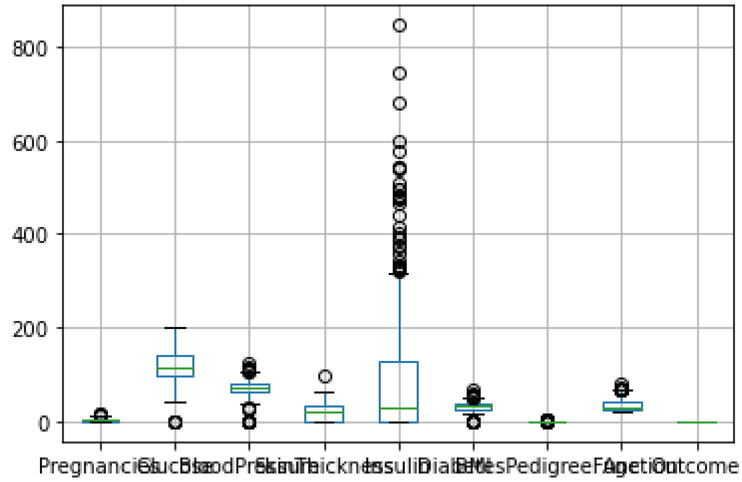
Out[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
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 [17]:
```
df.shape
```

Out[17]:
```
(768, 9)
```

In [18]:
```
df.boxplot();
```



In [19]:
```
df.isnull().sum()
```

Out[19]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [20]:
```
df.columns
```

Out[20]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

a value of zero does not make sense and thus indicates missing value:Treating the null values

In [21]:
```
for i in['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
         'BMI']:
    print(i)
    print(df[i].value_counts(normalize=True)[0], '\n\n')
```

```
Glucose
0.006510416666666667


BloodPressure
0.045572916666666664


SkinThickness
0.2955729166666667


Insulin
0.4869791666666667


BMI
```

```
0.014322916666666666
```

In [22]:

```
df[df['Insulin']!=0]['Insulin'].median()
```

Out[22]:

```
125.0
```

In [23]:

```
for i in['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI']:
    print(i)
    median_value=df[df[i]!=0][i].median()
    print(median_value)
    df[i].replace(0,median_value,inplace=True)
```

```
Glucose
117.0
BloodPressure
72.0
SkinThickness
29.0
Insulin
125.0
BMI
32.3
```

3. 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.

In [24]:

```
df.dtypes.value_counts()
```

Out[24]:

```
int64      7
float64    2
dtype: int64
```

In [25]:

```
df[df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI']]==0].count()
```

Out[25]:

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

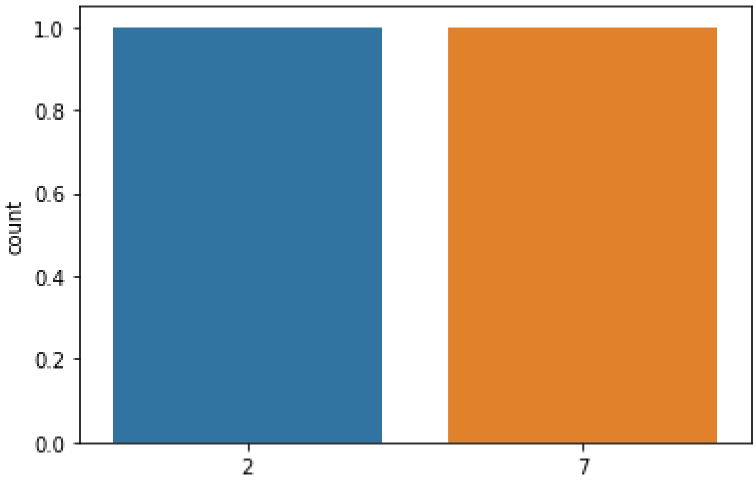
In [51]:

```
sns.countplot(df.dtypes.value_counts())
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[51]:

```
<AxesSubplot:ylabel='count'>
```



In [26]:

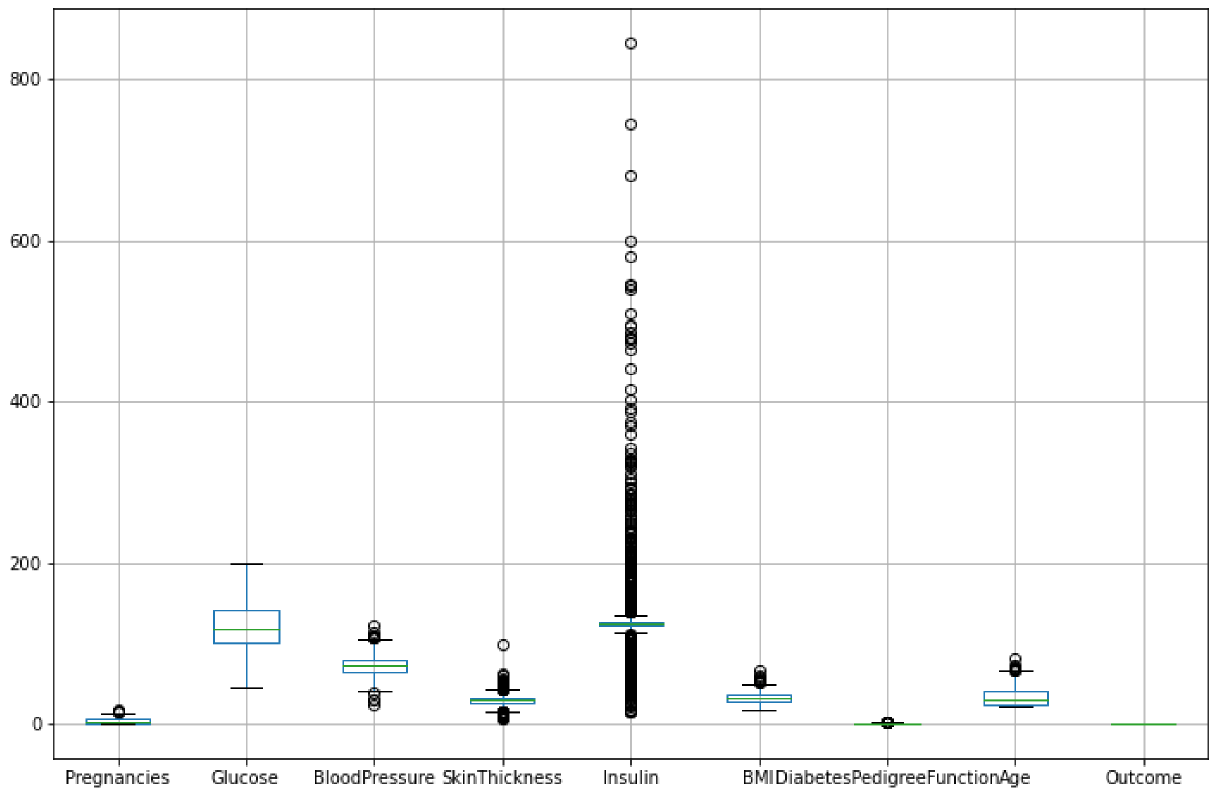
```
df.dtypes.value_counts()
```

Out[26]:

```
int64      7
float64    2
dtype: int64
```

In [27]:

```
plt.figure(figsize=(12,8))
df.boxplot();
```



In [28]:

```
df.columns
```

Out[28]:

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

Check the balance of the data by plotting the count of outcomes by their value.

In [29]:
```
df.Outcome.value_counts(normalize=True)
```

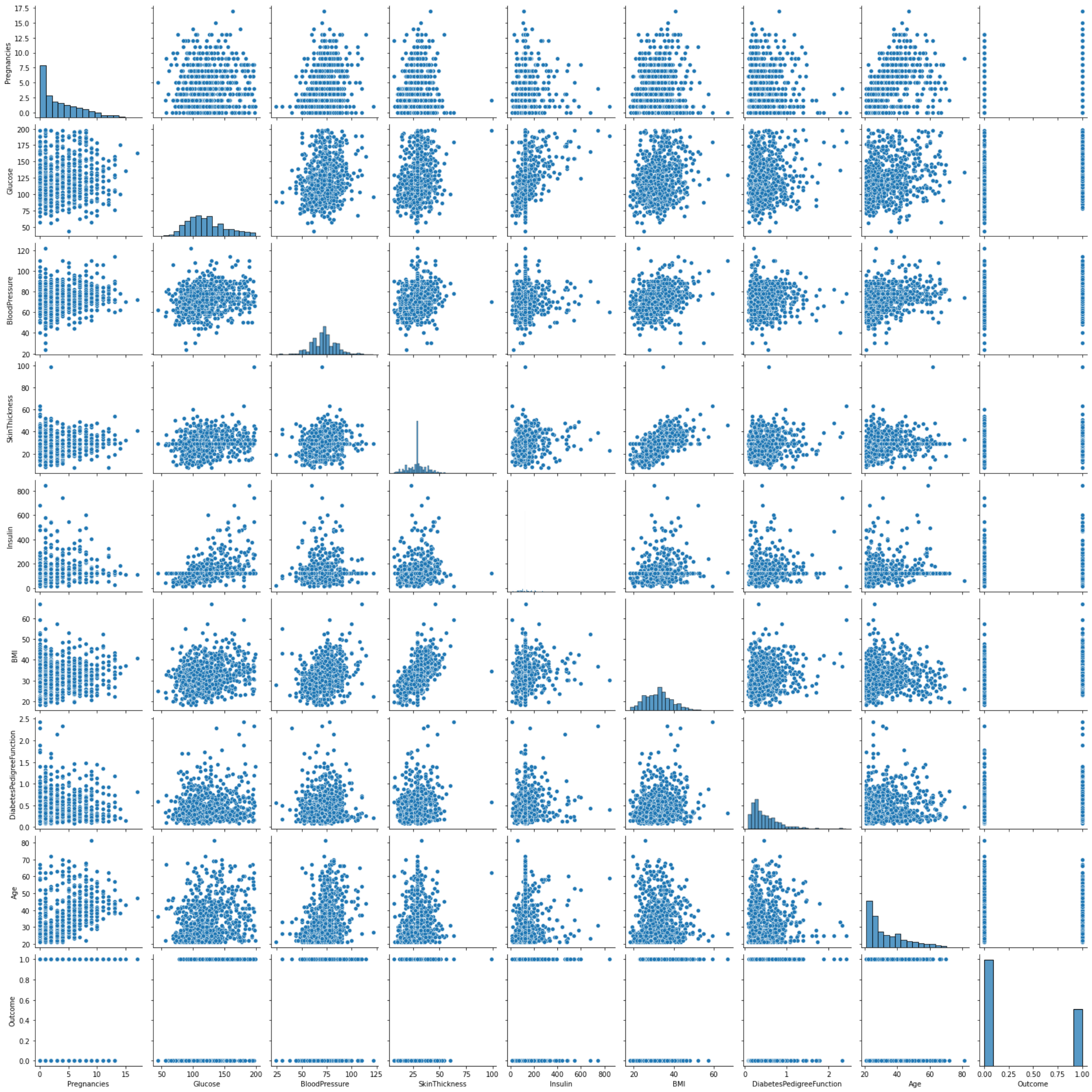
Out[29]:
```
0    0.651042
1    0.348958
Name: Outcome, dtype: float64
```

Create scatter charts between the pair of variables to understand the relationships.

Describe your findings.

In [30]:
```
sns.pairplot(df)
```

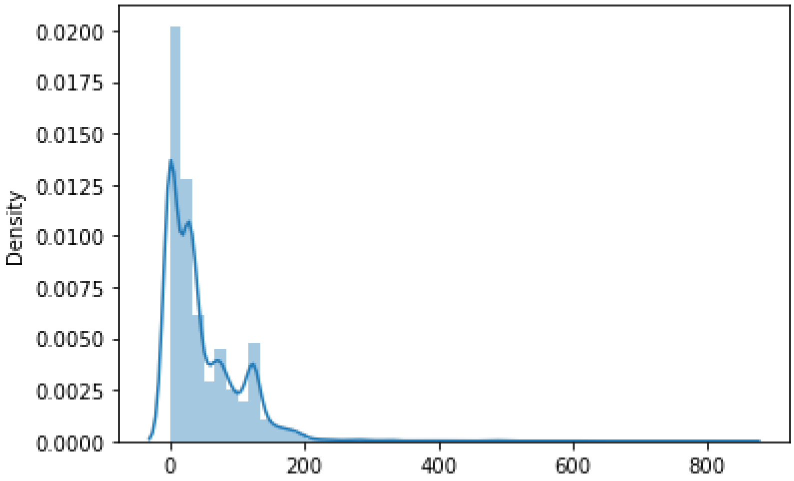
Out[30]:
```
<seaborn.axisgrid.PairGrid at 0x1f197d0cf40>
```



In [52]:
```
sns.distplot(df)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

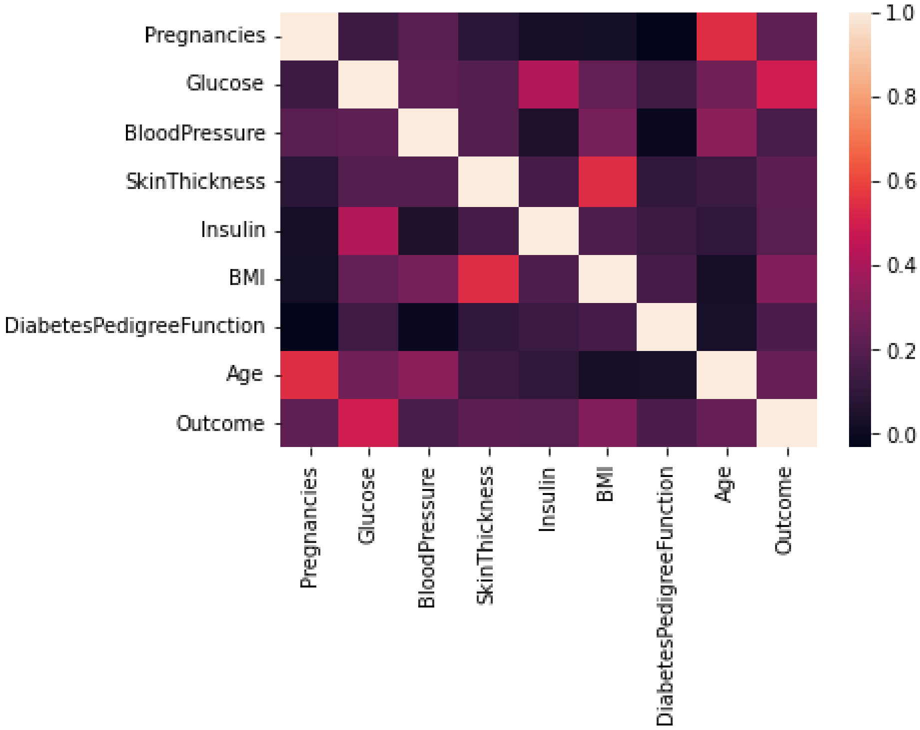
Out[52]:
```
<AxesSubplot:ylabel='Density'>
```



Perform correlation analysis. Visually explore it using a heat map.

In [31]:
```
sns.heatmap(df.corr())
```

Out[31]: <AxesSubplot:>



In [32]:
```
df.columns
```

Out[32]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [33]:
```
x=df.drop('Outcome',axis=1)
y=df['Outcome']
```

In [34]:
```
df['Outcome'].value_counts(normalize=True)
```

Out[34]:
```
0    0.651042
1    0.348958
Name: Outcome, dtype: float64
```

In [35]:
```
train_x,test_x,train_y,test_y=train_test_split(x,y,test_size=25,random_state=42,stra
```

In [36]:
```
train_x.shape
```

Out[36]:
```
(743, 8)
```

In [37]:
```
test_x.shape
```

Out[37]:
```
(25, 8)
```

In [38]:
```
test_y.value_counts(normalize=True)
```

Out[38]:
```
0    0.64
1    0.36
Name: Outcome, dtype: float64
```

In [39]:
```
train_y.value_counts(normalize=True)
```

Out[39]:
```
0    0.651413
1    0.348587
Name: Outcome, dtype: float64
```

In [40]:
```
model=KNeighborsClassifier()
```

In [41]:
```
model.fit(train_x,train_y)
```

Out[41]:
```
KNeighborsClassifier()
```

In [42]:
```
model.score(train_x,train_y)
```

Out[42]:
```
0.7981157469717362
```

In [43]:
```
model.score(test_x,test_y)
```

Out[43]:
```
0.76
```

Normalizing the Train and test data Using Minmax scalar

In [53]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [54]:
```
scalar=MinMaxScaler()
```

In [56]:
```
scaled_train_x=scalar.fit_transform(train_x)
```

In [57]:
```
scaled_test_x=scalar.fit_transform(test_x)
```

KNN classifier model

In [59]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [60]:
```
knn_model=KNeighborsClassifier(n_neighbors=20)
```

In [62]:
```
knn_model.fit(scaled_train_x,train_y)
```

Out[62]:
```
KNeighborsClassifier(n_neighbors=20)
```

In [63]:
```
pred_y_test_knn=knn_model.predict(scaled_test_x)
```

In [64]:
```
from sklearn.metrics import accuracy_score

print("The accuracy score of the model on test data is : ")
print(accuracy_score(test_y , pred_y_test_knn))
```

```
The accuracy score of the model on test data is :
0.84
```

In [65]:
```
pred_y_train_knn=knn_model.predict(scaled_train_x)
```

In [66]:
```
print("The accuracy score of the model on train data is : ")
print(accuracy_score(train_y , pred_y_train_knn))
```

```
The accuracy score of the model on train data is :
0.7860026917900403
```

We could infer that a higher K value would affect the prediction accuracy

In [45]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [46]:
```
y_pred=model.predict(test_x)
```

In [47]:
```
print(classification_report(test_y,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        16
           1       0.67      0.67      0.67         9

    accuracy                           0.76        25
   macro avg       0.74      0.74      0.74        25
weighted avg       0.76      0.76      0.76        25
```

In [48]:
```
print(confusion_matrix(test_y,y_pred))
```

```
[[13  3]
 [ 3  6]]
```

Naive Bayes Classifier - Gaussian NB

In [68]:
```
from sklearn.naive_bayes import GaussianNB
```

In [69]:
```
gnb_model=GaussianNB()
```

In [71]:
```
gnb_model.fit(scaled_train_x,train_y)
```

Out[71]:
```
GaussianNB()
```

In [73]:
```
y_gnb_test=gnb_model.predict(scaled_test_x)
```

In [74]:
```
print("The accuracy score of the model on test data is : ")
print(accuracy_score(test_y , y_gnb_test))
```

```
The accuracy score of the model on test data is :
0.68
```

In [75]:
```
y_gnb_train=gnb_model.predict(scaled_train_x)
```

In [78]:
```
print(classification_report(test_y , y_gnb_test))
```

```
              precision    recall  f1-score   support

           0       0.90      0.56      0.69        16
           1       0.53      0.89      0.67         9

    accuracy                           0.68        25
   macro avg       0.72      0.73      0.68        25
weighted avg       0.77      0.68      0.68        25
```

Build SVC Model

In [79]:
```
from sklearn.svm import SVC
```

In [80]:
```
svc_model=SVC()
```

In [82]:
```
svc_model.fit(scaled_train_x,train_y)
```

Out[82]:
```
SVC()
```

In [83]:
```
svc_model_test_pred=svc_model.predict(scaled_test_x)
```

In [85]:
```
svc_model_train_pred=svc_model.predict(scaled_train_x)
```

In [87]:
```
print("The accuracy score of the model on test data is : ")
print(accuracy_score(test_y , svc_model_test_pred))
```

```
The accuracy score of the model on test data is :
0.6
```

In [88]:

```
print("The accuracy score of the model on train data is : ")
print(accuracy_score(train_y , svc_model_train_pred))
```

```
The accuracy score of the model on train data is :
0.8088829071332436
```

In [90]:

```
print(classification_report(test_y , svc_model_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.67      0.75      0.71        16
           1       0.43      0.33      0.38         9

    accuracy                           0.60        25
   macro avg       0.55      0.54      0.54        25
weighted avg       0.58      0.60      0.59        25
```

In [ ]: