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

In [2]:
```
df=pd.read_csv("data.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoc
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

5 rows × 33 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [5]:
```
df.isna().sum()
```

Out[5]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32                569
dtype: int64
```

In [6]:
```
df.shape
```

Out[6]:
```
(569, 33)
```

In [7]:
```
df=df.dropna(axis=1)
```

In [8]:
```
df.shape
```

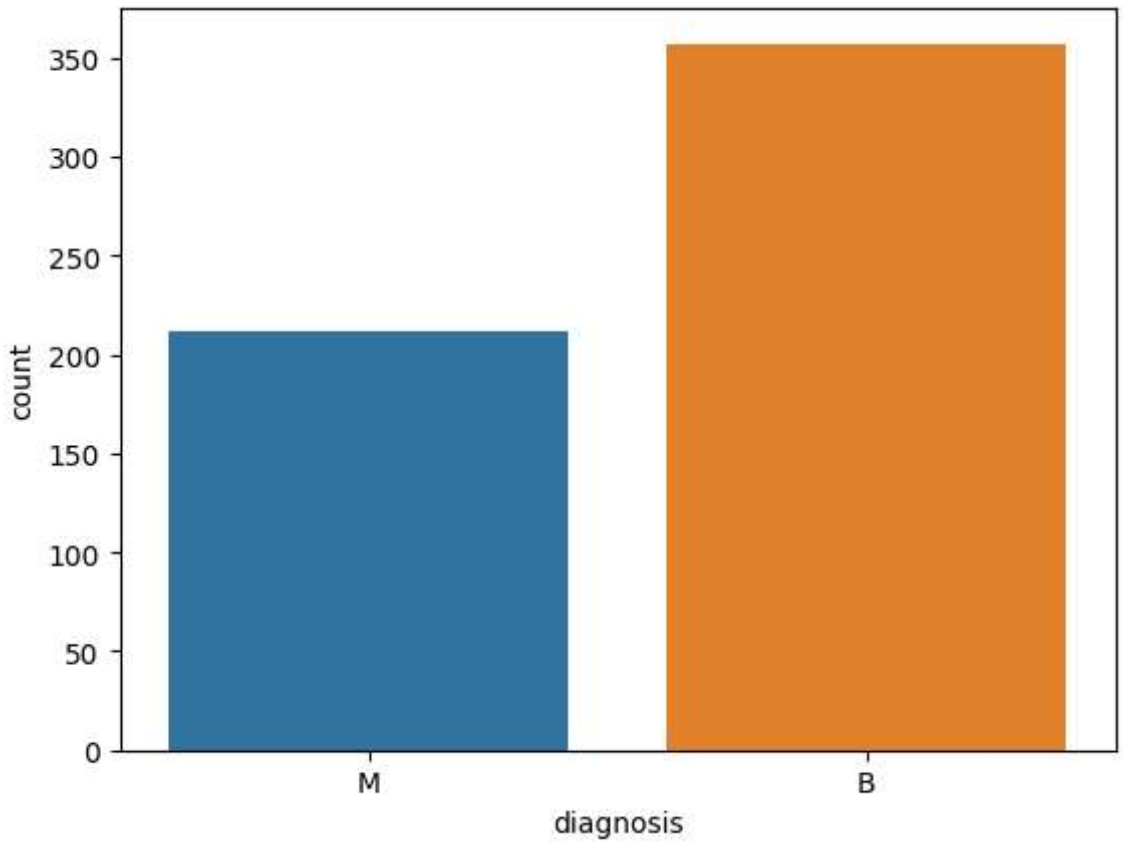
Out[8]:
```
(569, 32)
```

In [9]:
```
df['diagnosis'].value_counts()
```

Out[9]:
```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

In [10]:
```
sns.countplot(x=df['diagnosis'])
```

Out[10]:
```
<Axes: xlabel='diagnosis', ylabel='count'>
```



In [11]:
```
# LABEL ENCODING (CONVERT THE VALUE OF M AND B INTO 1 AND 0)
from sklearn.preprocessing import LabelEncoder
labelencoder_Y = LabelEncoder()
df.iloc[:,1]=labelencoder_Y.fit_transform(df.iloc[:,1].values)
```

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

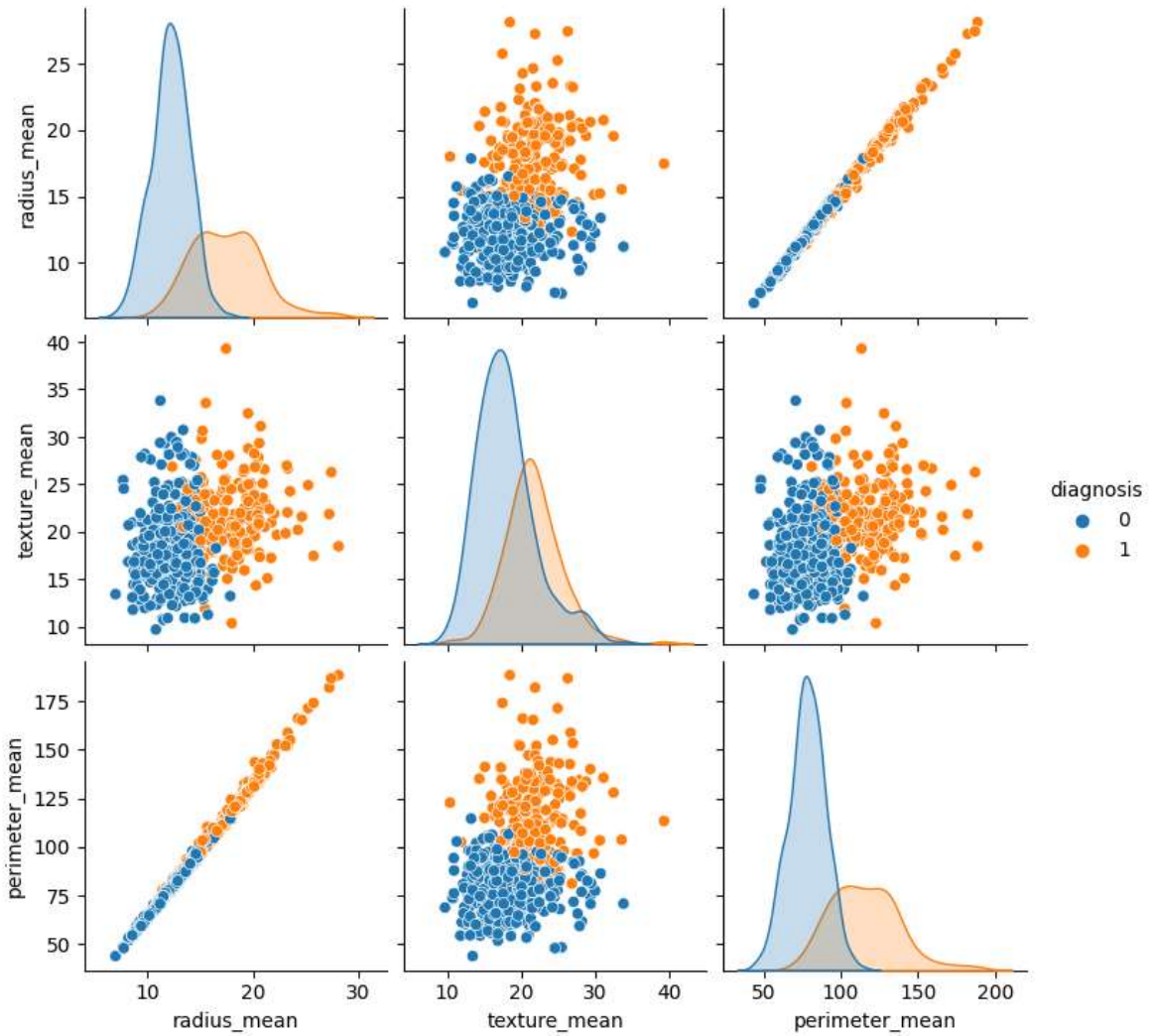
Out[12]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoc
0	842302	1	17.99	10.38	122.80	1001.0	
1	842517	1	20.57	17.77	132.90	1326.0	
2	84300903	1	19.69	21.25	130.00	1203.0	
3	84348301	1	11.42	20.38	77.58	386.1	
4	84358402	1	20.29	14.34	135.10	1297.0	

5 rows × 32 columns

In [13]:
```
sns.pairplot(df.iloc[:,1:5], hue="diagnosis")
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x17effc70750>
```



In [14]:
```
# GET THE CORRELATION
df.iloc[:,1:32].corr()
```

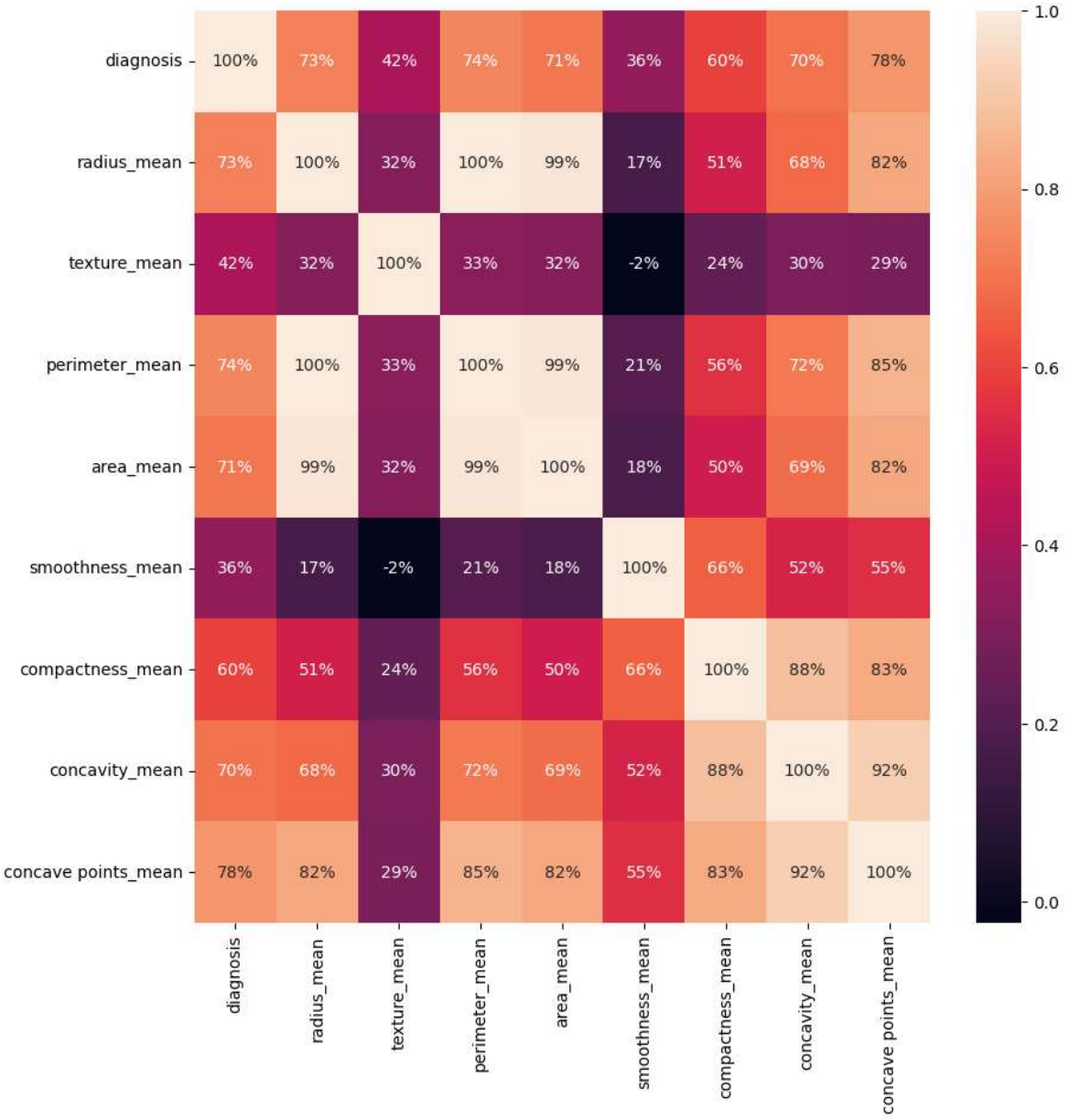
Out[14]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_n
diagnosis	1.000000	0.730029	0.415185	0.742636	0.70
radius_mean	0.730029	1.000000	0.323782	0.997855	0.98
texture_mean	0.415185	0.323782	1.000000	0.329533	0.32
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.98
area_mean	0.708984	0.987357	0.321086	0.986507	1.00
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.17
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.49
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.68
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.82
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.15
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.28
radius_se	0.567134	0.679090	0.275869	0.691765	0.73
texture_se	-0.008303	-0.097317	0.386358	-0.086761	-0.06
perimeter_se	0.556141	0.674172	0.281673	0.693135	0.72
area_se	0.548236	0.735864	0.259845	0.744983	0.80
smoothness_se	-0.067016	-0.222600	0.006614	-0.202694	-0.16
compactness_se	0.292999	0.206000	0.191975	0.250744	0.21
concavity_se	0.253730	0.194204	0.143293	0.228082	0.20
concave points_se	0.408042	0.376169	0.163851	0.407217	0.37
symmetry_se	-0.006522	-0.104321	0.009127	-0.081629	-0.07
fractal_dimension_se	0.077972	-0.042641	0.054458	-0.005523	-0.01
radius_worst	0.776454	0.969539	0.352573	0.969476	0.96
texture_worst	0.456903	0.297008	0.912045	0.303038	0.28
perimeter_worst	0.782914	0.965137	0.358040	0.970387	0.95
area_worst	0.733825	0.941082	0.343546	0.941550	0.95
smoothness_worst	0.421465	0.119616	0.077503	0.150549	0.12
compactness_worst	0.590998	0.413463	0.277830	0.455774	0.39
concavity_worst	0.659610	0.526911	0.301025	0.563879	0.51
concave points_worst	0.793566	0.744214	0.295316	0.771241	0.72
symmetry_worst	0.416294	0.163953	0.105008	0.189115	0.14
fractal_dimension_worst	0.323872	0.007066	0.119205	0.051019	0.00

31 rows × 31 columns

In [15]:
```
# visualise the correlation
plt.figure(figsize=(10,10))
sns.heatmap(df.iloc[:,1:10].corr(),annot=True ,fmt='.0%')
```

Out[15]: <Axes: >



In [16]:
```
# SPLIT THE DATASET INTO DEPENDENT(X) AND INDEPENDENT(Y) DATASETS
X=df.iloc[:,2:31].values
Y=df.iloc[:,1].astype('int').values
```

In [17]:
```
# NOW IT IS NOT A DATAFRAME , IT BECOME LIST OF LIST THAT IS WHY USE PRINT
print(X)
```

```
[[ 17.99    10.38   122.8    ...   0.7119   0.2654   0.4601]
 [ 20.57    17.77   132.9    ...   0.2416   0.186    0.275 ]
 [ 19.69    21.25   130.     ...   0.4504   0.243    0.3613]
 ...
 [ 16.6     28.08   108.3    ...   0.3403   0.1418   0.2218]
 [ 20.6     29.33   140.1    ...   0.9387   0.265    0.4087]
 [  7.76    24.54    47.92   ...   0.       0.       0.2871]]
```

In [18]:
```
print(Y)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1
 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0
 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 1 0 1
 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 1 0 0
 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0
 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1
 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0]
```

In [19]:
```
# SPLITING THE DATA INTO TRAINING AND TESTING DATASETS
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
```

In [20]:
```
# FEATURE SCALING
from sklearn.preprocessing import StandardScaler
X_train=StandardScaler().fit_transform(X_train)
X_test=StandardScaler().fit_transform(X_test)
```

In [21]:
```
def models(X_train,Y_train):

    # LOGISTIC REGRESSION
    from sklearn.linear_model import LogisticRegression
    log=LogisticRegression(random_state=0)
    log.fit(X_train,Y_train)


    # DECISION TREEE
    from sklearn.tree import DecisionTreeClassifier
    tree=DecisionTreeClassifier(random_state=0,criterion="entropy")
    tree.fit(X_train,Y_train)


    # RANDOM FOREST
    from sklearn.ensemble import RandomForestClassifier
    forest=RandomForestClassifier(random_state=0,criterion="entropy",n_estimator
    forest.fit(X_train,Y_train)


    # k-nearest neighbour
    from sklearn.neighbors import KNeighborsClassifier
    knn=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
    knn.fit(X_train,Y_train)


    #NAIVE BAyes
    from sklearn.naive_bayes import GaussianNB
    nb=GaussianNB()
    nb.fit(X_train,Y_train)
```

```
    print('[0]logistic regression accuracy:',log.score(X_train,Y_train))
    print('[1]Decision tree accuracy:',tree.score(X_train,Y_train))
    print('[2]Random forest accuracy:',forest.score(X_train,Y_train))
    print('[3]K-nearest neighbor accuracy:',knn.score(X_train,Y_train))
    print('[4]Naive bayes accuracy:',nb.score(X_train,Y_train))


    return log,tree,forest,knn,nb
```

In [22]:
```
model=models(X_train,Y_train)
```

```
[0]logistic regression accuracy: 0.9912087912087912
[1]Decision tree accuracy: 1.0
[2]Random forest accuracy: 0.9978021978021978
[3]K-nearest neighbor accuracy: 0.9802197802197802
[4]Naive bayes accuracy: 0.9494505494505494
```

In [23]:
```
#TESTING THE MODEL/RESULT

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

for i in range(len(model)):
    print("Model",i)
    print(classification_report(Y_test,model[i].predict(X_test)))
    print('Accuracy :',accuracy_score(Y_test,model[i].predict(X_test)))
```

```
Model 0
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        67
           1       0.98      0.94      0.96        47

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114

Accuracy : 0.9649122807017544
Model 1
              precision    recall  f1-score   support

           0       0.94      0.96      0.95        67
           1       0.93      0.91      0.92        47

    accuracy                           0.94       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.94      0.94      0.94       114

Accuracy : 0.9385964912280702
Model 2
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        67
           1       1.00      0.94      0.97        47

    accuracy                           0.97       114
   macro avg       0.98      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

Accuracy : 0.9736842105263158
Model 3
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        67
           1       1.00      0.91      0.96        47

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114

Accuracy : 0.9649122807017544
Model 4
              precision    recall  f1-score   support

           0       0.94      0.96      0.95        67
           1       0.93      0.91      0.92        47

    accuracy                           0.94       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.94      0.94      0.94       114

Accuracy : 0.9385964912280702
```

In [24]:
```
#PREDICTION OF RANDOM FOREST
pred=model[2].predict(X_test)
print("Predicted Values:")
print(pred)
```

```
print("Actual Values:")
print(Y_test)
```

```
Predicted Values:
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0
 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0
 1 1 0]
Actual Values:
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1
 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0
 1 1 0]
```

In [25]:
```
from joblib import dump
dump(model[2],"Breast_Cancer_Detection_Model.joblib")
```

Out[25]:
```
['Breast_Cancer_Detection_Model.joblib']
```

In [ ]:
```

```