Importing requied libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Reading and importing Dataset which is Breast cancer detection

```
df=pd.read_csv("breast-cancer.csv.xls")
```

```
df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
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	
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

569 rows × 32 columns

checking descriptive statistics

```
df.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactnes
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0

8 rows × 31 columns

checking total number of rows and columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 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
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

Checking columns(features)

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactn
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 32 columns

```
df["diagnosis"].value_counts()
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

^ according to description of dataset B means BENIGN not cancerous and M means MALIGNANT it is cancerous

in this dataset ID columns in not required

```
df.drop("id",inplace=True,axis=1)
```

```
df
```

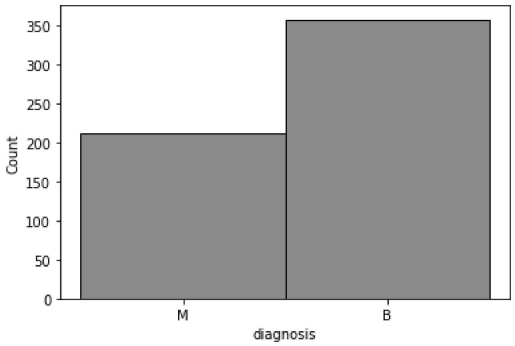
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280
...	...	...	...	...	...	...	..
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362

569 rows × 31 columns

Visualization

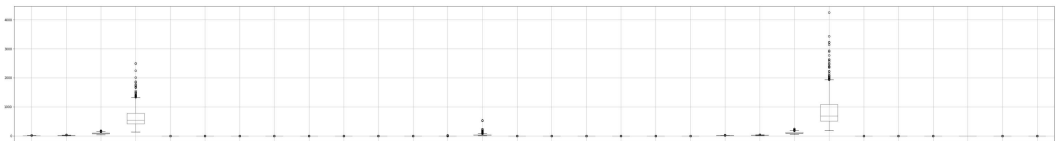
```
histplot=sns.histplot(df["diagnosis"],label="count")
B,M = df["diagnosis"].value_counts()
print("number of Benign(not cancerous): ",B)
print("number of Malignant(cancerous): ",M)
```

```
number of Benign(not cancerous):  357
number of Malignant(cancerous):  212
```



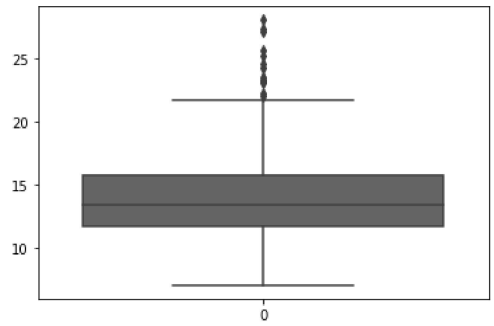

```
plt.figure(figsize=(60,8))
df.boxplot()
```

```
<Axes: >
```



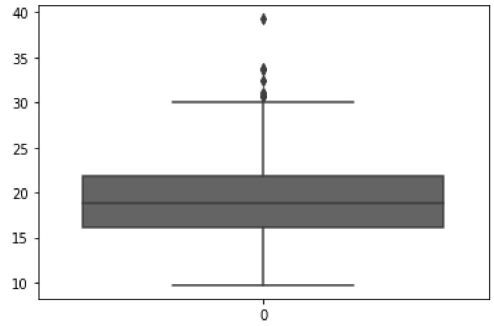
```
sns.boxplot(df["radius_mean"])
```

```
<Axes: >
```



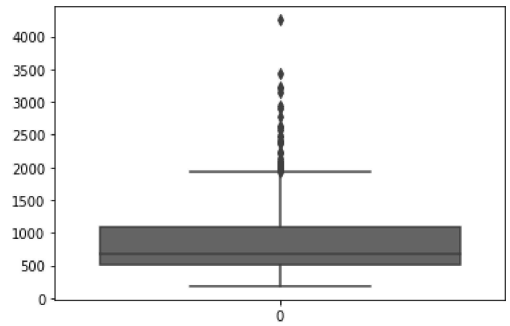

```
sns.boxplot(df["texture_mean"])
```

```
<Axes: >
```



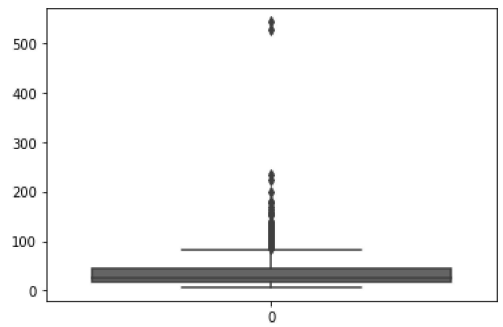

```
sns.boxplot(df["area_worst"])
```

```
<Axes: >
```




```
sns.boxplot(df["area_se"])
```

```
<Axes: >
```



```
#THERE ARE SOME OUTLIERS PRESENT IN BOXPLOTS
```

REMOVING OUTLIERS

```
df=df[(df["radius_mean"] < 23) & (df["texture_mean"] < 35) & (df["area_worst"] < 2300)  & (df["area_se"] < 150)]
```

```
df
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280
5	M	12.45	15.70	82.57	477.1	0.12780	0.17000
...	...	...	...	...	...	...	..
563	M	20.92	25.09	143.00	1347.0	0.10990	0.22360
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362

546 rows × 31 columns

so accourding to histplot there are 200 cancerous patients and 350 non cancerous so the count of non cancerous patients are greater than cancerous patients

splitting data into x(features) and y(label)

```
x=df.iloc[:,1:]
```

```
x
```

```
          radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  compactness_mean  concavity
```

```
y=df.iloc[:,:1].values
```

```
y
```

```
array([['M'],
       ['B'],
       ['B'],
       ['M'],
       ['B'],
       ['M'],
       ['B'],
       ['M'],
       ['M'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['M'],
       ['B'],
       ['M'],
       ['M'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['B'],
       ['M'],
       ['M'],
       ['M'],
       ['M'],
       ['M'],
       ['B']], dtype=object)
```

```
# Encoding for label
```

ENCODING CATEGORICAL LABEL (Y)

converting categorical label into numeric

```
from sklearn.preprocessing import LabelEncoder
```

```
le=LabelEncoder()
```

```
y=le.fit_transform(y)
```

```
y
```

```
array([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, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 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, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 1, 0, 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, 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, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 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, 0, 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, 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, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 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, 0])
```

1 = M (cancerous) , 0 = B (non cancerous)

performing standardiation(scailing) for better results

```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
```

```
x=sc.fit_transform(x)
```

```
x
```

```
array([[ 2.21677361, -0.32770195,  2.06017605, ...,  1.22694486,
         -0.239839  ,  0.27945786],
       [ 1.92977419,  0.51032166,  1.92249255, ...,  2.14473916,
         1.15564227,  0.20033731],
       [-0.76736803,  0.30081576, -0.56625533, ...,  2.37821315,
         6.04710326,  4.89372479],
       ...,
       [ 0.92201489,  2.15506342,  0.89224019, ...,  0.51525173,
        -1.10008932, -0.31504521],
       [ 2.22655768,  2.45607765,  2.40201094, ...,  2.49897556,
         1.92210591,  2.20142806],
       [-1.96102468,  1.30259112, -1.97442516, ..., -1.76796287,
        -0.04418056, -0.74416434]])
```

SPLITTING DATA INTO TRAINING AND TESTING

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

BUILDING A FUNCTION FOR MODELS AND FOR ACCURACY

```
def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)

    train=model.score(xtrain,ytrain)
    test=model.score(xtest,ytest)

    print(f"Training Acuuracy(bias):- {train}\n Testing Accuracy(variance):- {test}")
    print(classification_report(ytest,ypred))

    return model
```

IMPORTING ALL MODELS WHICH ARE REQUIRED

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import classification_report


# FOR LOGISTIC REGRESSION


  logreg = mymodel(LogisticRegression())
```

```
    Training Acuuracy(bias):- 0.9842931937172775
     Testing Accuracy(variance):- 0.9939024390243902
                  precision    recall  f1-score   support

               0       1.00      0.99      1.00       112
               1       0.98      1.00      0.99        52

        accuracy                           0.99       164
       macro avg       0.99      1.00      0.99       164
    weighted avg       0.99      0.99      0.99       164
```

```
# FOR KNN ALGORITHM


k = mymodel(KNeighborsClassifier())
```

```
    Training Acuuracy(bias):- 0.9790575916230366
     Testing Accuracy(variance):- 0.9817073170731707
                  precision    recall  f1-score   support

               0       0.99      0.98      0.99       112
               1       0.96      0.98      0.97        52

        accuracy                           0.98       164
       macro avg       0.98      0.98      0.98       164
    weighted avg       0.98      0.98      0.98       164
```

```
# FOR SUPPORT VECTOR MACHINE


s = mymodel(SVC())
```

```
    Training Acuuracy(bias):- 0.9869109947643979
     Testing Accuracy(variance):- 0.975609756097561
                  precision    recall  f1-score   support

               0       0.98      0.98      0.98       112
               1       0.96      0.96      0.96        52

        accuracy                           0.98       164
       macro avg       0.97      0.97      0.97       164
    weighted avg       0.98      0.98      0.98       164
```

```
#FOR  DECISION TRESS


d = mymodel(DecisionTreeClassifier())

     Training Acuuracy(bias):- 1.0
      Testing Accuracy(variance):- 0.9207317073170732
                  precision    recall  f1-score   support

               0       0.96      0.92      0.94       112
               1       0.84      0.92      0.88        52

        accuracy                           0.92       164
       macro avg       0.90      0.92      0.91       164
    weighted avg       0.92      0.92      0.92       164



#FOR RANDOM FOREST


r = mymodel(RandomForestClassifier())

     Training Acuuracy(bias):- 1.0
      Testing Accuracy(variance):- 0.9634146341463414
                  precision    recall  f1-score   support

               0       0.98      0.96      0.97       112
               1       0.93      0.96      0.94        52

        accuracy                           0.96       164
       macro avg       0.95      0.96      0.96       164
    weighted avg       0.96      0.96      0.96       164



# FOR BAYES THEORM


n = mymodel(BernoulliNB())

     Training Acuuracy(bias):- 0.9240837696335078
      Testing Accuracy(variance):- 0.9451219512195121
                  precision    recall  f1-score   support

               0       0.99      0.93      0.96       112
               1       0.86      0.98      0.92        52

        accuracy                           0.95       164
       macro avg       0.93      0.95      0.94       164
    weighted avg       0.95      0.95      0.95       164
```

OBSERVING ALL THE MODELS ACCURACY OF LOGISTIC REGRESSION IS BEST FOR THIS DATASET BECAUSE OF IT IS A BINARY CLASSSIFIACTION DATASET

WHAT IF WE USE ARTIFICIAL NEURAl NETWORK ON THIS DATASET LETS CHECK

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


from tensorflow.keras.callbacks import EarlyStopping


early_stop= EarlyStopping(monitor="val_loss",mode="min",verbose=1,patience=10)


model=Sequential()
model.add(Dense(20,activation="relu"))
model.add(Dense(20,activation="relu"))
model.add(Dense(1,activation="sigmoid"))
```

```
model.compile(optimizer="sgd",loss="binary_crossentropy",metrics=["accuracy"])
```

```
model.fit(xtrain,ytrain,epochs=600,validation_data=(xtest,ytest),verbose=1,batch_size=30,callbacks=[early_stop])
```

```
Epoch 87/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0762 - accuracy: 0.9843 - val_loss: 0.0435 - val_accuracy: 0.9878
Epoch 88/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0758 - accuracy: 0.9843 - val_loss: 0.0432 - val_accuracy: 0.9878
Epoch 89/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0753 - accuracy: 0.9843 - val_loss: 0.0428 - val_accuracy: 0.9878
Epoch 90/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0748 - accuracy: 0.9843 - val_loss: 0.0425 - val_accuracy: 0.9878
Epoch 91/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0744 - accuracy: 0.9843 - val_loss: 0.0422 - val_accuracy: 0.9878
Epoch 92/600
13/13 [==============================] - 0s 5ms/step - loss: 0.0740 - accuracy: 0.9843 - val_loss: 0.0419 - val_accuracy: 0.9878
Epoch 93/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0735 - accuracy: 0.9843 - val_loss: 0.0416 - val_accuracy: 0.9878
Epoch 94/600
13/13 [==============================] - 0s 6ms/step - loss: 0.0732 - accuracy: 0.9843 - val_loss: 0.0413 - val_accuracy: 0.9878
Epoch 95/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0727 - accuracy: 0.9843 - val_loss: 0.0410 - val_accuracy: 0.9878
Epoch 96/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0723 - accuracy: 0.9869 - val_loss: 0.0407 - val_accuracy: 0.9878
Epoch 97/600
13/13 [==============================] - 0s 5ms/step - loss: 0.0719 - accuracy: 0.9817 - val_loss: 0.0404 - val_accuracy: 0.9878
Epoch 98/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0715 - accuracy: 0.9843 - val_loss: 0.0401 - val_accuracy: 0.9878
Epoch 99/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0711 - accuracy: 0.9817 - val_loss: 0.0399 - val_accuracy: 0.9878
Epoch 100/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0707 - accuracy: 0.9869 - val_loss: 0.0396 - val_accuracy: 0.9878
Epoch 101/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0704 - accuracy: 0.9869 - val_loss: 0.0394 - val_accuracy: 0.9878
Epoch 102/600
13/13 [==============================] - 0s 5ms/step - loss: 0.0700 - accuracy: 0.9843 - val_loss: 0.0392 - val_accuracy: 0.9878
Epoch 103/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0696 - accuracy: 0.9843 - val_loss: 0.0390 - val_accuracy: 0.9878
Epoch 104/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0692 - accuracy: 0.9843 - val_loss: 0.0388 - val_accuracy: 0.9878
Epoch 105/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0688 - accuracy: 0.9843 - val_loss: 0.0385 - val_accuracy: 0.9878
Epoch 106/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0685 - accuracy: 0.9843 - val_loss: 0.0383 - val_accuracy: 0.9878
Epoch 107/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0681 - accuracy: 0.9843 - val_loss: 0.0380 - val_accuracy: 0.9878
Epoch 108/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0679 - accuracy: 0.9843 - val_loss: 0.0379 - val_accuracy: 0.9878
Epoch 109/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0673 - accuracy: 0.9843 - val_loss: 0.0377 - val_accuracy: 0.9878
Epoch 110/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0670 - accuracy: 0.9843 - val_loss: 0.0374 - val_accuracy: 0.9878
Epoch 111/600
13/13 [==============================] - 0s 5ms/step - loss: 0.0666 - accuracy: 0.9843 - val_loss: 0.0372 - val_accuracy: 0.9878
Epoch 112/600
13/13 [==============================] - 0s 5ms/step - loss: 0.0662 - accuracy: 0.9843 - val_loss: 0.0370 - val_accuracy: 0.9878
Epoch 113/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0660 - accuracy: 0.9843 - val_loss: 0.0369 - val_accuracy: 0.9878
Epoch 114/600
13/13 [==============================] - 0s 5ms/step - loss: 0.0656 - accuracy: 0.9843 - val_loss: 0.0366 - val_accuracy: 0.9878
Epoch 115/600
13/13 [==============================] - 0s 4ms/step - loss: 0.0651 - accuracy: 0.9843 - val_loss: 0.0364 - val_accuracy: 0.9878
Epoch 116/600
```

```
ypred=model.predict(xtest)
```

```
6/6 [==============================] - 0s 1ms/step
```

```
ypred=ypred>0.5
```

```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       112
           1       1.00      1.00      1.00        52

    accuracy                           1.00       164
   macro avg       1.00      1.00      1.00       164
weighted avg       1.00      1.00      1.00       164
```

OUR ACCURACY OF ARTIFICIAL NEURAL NETWORK IS 100 PERCENT

✓ 0s completed at 8:25 PM