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

In [3]:
```
df=pd.read_csv("C:/Users/elise/Downloads/dataset.csv")
```

In [4]:
```
df.head(10)
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003
5	843786	M	12.45	15.70	82.57	477.1	0.1278
6	844359	M	18.25	19.98	119.60	1040.0	0.0946
7	84458202	M	13.71	20.83	90.20	577.9	0.1189
8	844981	M	13.00	21.82	87.50	519.8	0.1273
9	84501001	M	12.46	24.04	83.97	475.9	0.1186

10 rows × 32 columns

In [5]:
```
# count the number of rows and columns in dataset:
df.shape
```

Out[5]:
```
(569, 32)
```

In [6]:
```
# count the number of empty values in each columns:
df.isna().sum()
```

Out[6]:
```
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
dtype: int64
```

In [7]:
```
# drop the columns with all the missing values:
df = df.dropna(axis = 1)
```

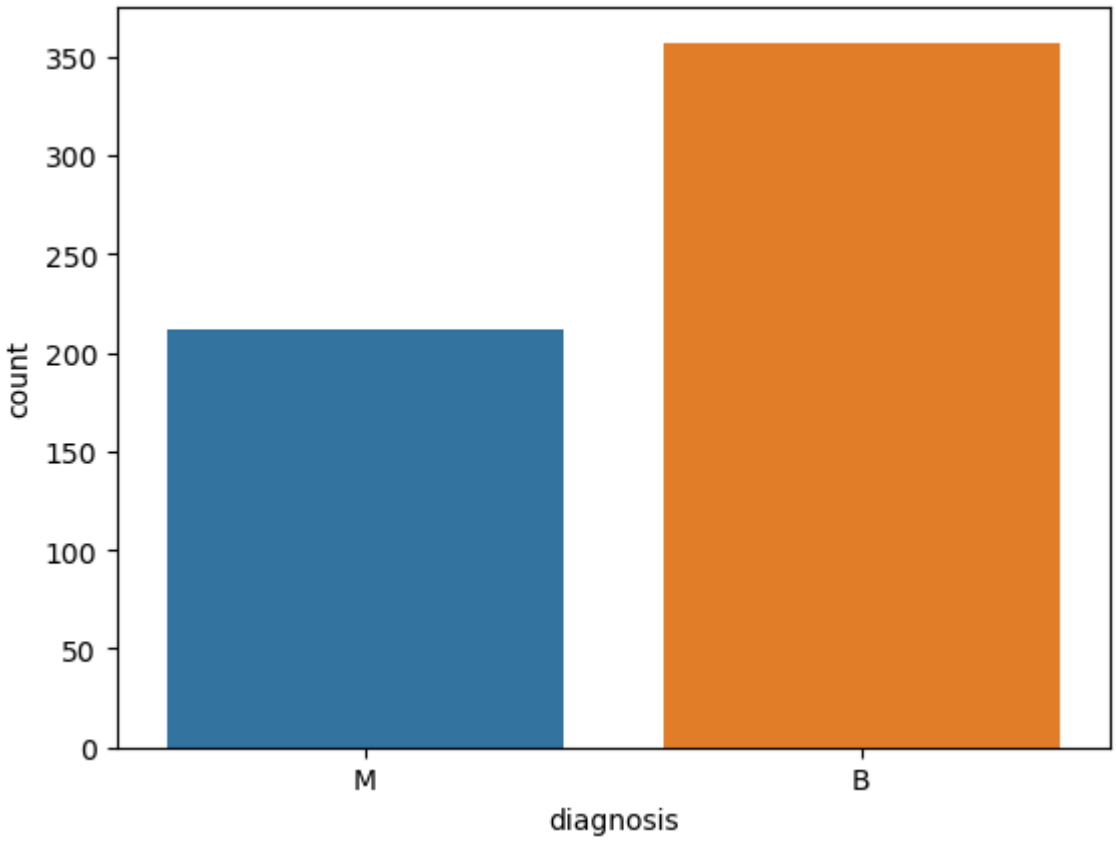
In [8]:
```
df.shape
```

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

In [9]:
```
# Get the count of the number of Malognant(M) or Benign(B) cells
df['diagnosis'].value_counts()
```

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

In [12]:
```
# visualize the count:
sns.countplot(data=df, x='diagnosis', label='count')
plt.show()
```



In [13]:
```
# look at the data types to see which columns need to be encoded:
df.dtypes
```

Out[13]:
```
id                         int64
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
```

In [14]:
```
# Rename the dagnosis data to labels:
df = df.rename(columns = {'diagnosis' : 'label'})
print(df.dtypes)
```

```
id                         int64
label                     object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
dtype: object
```

In [15]:
```
# define the dependent variable that need to predict(label)
y = df['label'].values
print(np.unique(y))
```

```
['B' 'M']
```

In [16]:
```
# Encoding categorical data from text(B and M) to integers (0 and 1)
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
Y = labelencoder.fit_transform(y) # M = 1 and B = 0
print(np.unique(Y))
```

```
[0 1]
```

In [17]:
```
# define x and normalize / scale value:

# define the independent variables, Drop label and ID , and normalize other data:
X  = df.drop(labels=['label','id'],axis = 1)

#scale / normalize the values to bring them into similar range:
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X = scaler.transform(X)
```

```
[[0.52103744 0.0226581  0.54598853 ... 0.91202749 0.59846245 0.41886396]
 [0.64314449 0.27257355 0.61578329 ... 0.63917526 0.23358959 0.22287813]
 [0.60149557 0.3902604  0.59574321 ... 0.83505155 0.40370589 0.21343303]
 ...
 [0.45525108 0.62123774 0.44578813 ... 0.48728522 0.12872068 0.1519087 ]
 [0.64456434 0.66351031 0.66553797 ... 0.91065292 0.49714173 0.45231536]
 [0.03686876 0.50152181 0.02853984 ... 0.         0.25744136 0.10068215]]
```

In [18]:
```
# Split data into training and testing data to verify accuracy after fitting the mc
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,Y, test_size = 0.25, random_stat
print('Shape of training data is: ', x_train.shape)
print('Shape of testing data is: ', x_test.shape)
```

```
Shape of training data is:  (426, 30)
Shape of testing data is:  (143, 30)
```

In [23]:
```
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
```

In [24]:
```
model = Sequential()
model.add(Dense(128, input_dim=30, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64,activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss = 'binary_crossentropy', optimizer = 'adam' , metrics = ['accura
```

In [22]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               3968

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

 activation (Activation)     (None, 1)                 0

=================================================================
Total params: 12289 (48.00 KB)
Trainable params: 12289 (48.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [25]:
```
# fit with no early stopping or other callbacks:
history = model.fit(x_train,y_train,verbose = 1,epochs = 100, batch_size = 64,valic
```

```
Epoch 1/100
WARNING:tensorflow:From C:\Users\elise\anaconda3\Lib\site-packages\keras\src\utils
\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use t
f.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\elise\anaconda3\Lib\site-packages\keras\src\engin
e\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is depr
ecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

7/7 [==============================] - 3s 91ms/step - loss: 0.6781 - accuracy: 0.5
587 - val_loss: 0.6486 - val_accuracy: 0.7762
Epoch 2/100
7/7 [==============================] - 0s 22ms/step - loss: 0.6498 - accuracy: 0.6
455 - val_loss: 0.6161 - val_accuracy: 0.8881
Epoch 3/100
7/7 [==============================] - 0s 19ms/step - loss: 0.6271 - accuracy: 0.7
019 - val_loss: 0.5785 - val_accuracy: 0.9231
Epoch 4/100
7/7 [==============================] - 0s 25ms/step - loss: 0.5869 - accuracy: 0.7
817 - val_loss: 0.5325 - val_accuracy: 0.9231
Epoch 5/100
7/7 [==============================] - 0s 27ms/step - loss: 0.5531 - accuracy: 0.8
146 - val_loss: 0.4786 - val_accuracy: 0.9371
Epoch 6/100
7/7 [==============================] - 0s 21ms/step - loss: 0.5017 - accuracy: 0.8
545 - val_loss: 0.4219 - val_accuracy: 0.9371
Epoch 7/100
7/7 [==============================] - 0s 20ms/step - loss: 0.4563 - accuracy: 0.8
521 - val_loss: 0.3611 - val_accuracy: 0.9301
Epoch 8/100
7/7 [==============================] - 0s 29ms/step - loss: 0.4236 - accuracy: 0.8
451 - val_loss: 0.3085 - val_accuracy: 0.9231
Epoch 9/100
7/7 [==============================] - 0s 17ms/step - loss: 0.3720 - accuracy: 0.8
638 - val_loss: 0.2678 - val_accuracy: 0.9371
Epoch 10/100
7/7 [==============================] - 0s 19ms/step - loss: 0.3373 - accuracy: 0.8
920 - val_loss: 0.2333 - val_accuracy: 0.9301
Epoch 11/100
7/7 [==============================] - 0s 21ms/step - loss: 0.3128 - accuracy: 0.8
897 - val_loss: 0.2127 - val_accuracy: 0.9441
Epoch 12/100
7/7 [==============================] - 0s 26ms/step - loss: 0.2745 - accuracy: 0.8
991 - val_loss: 0.1886 - val_accuracy: 0.9441
Epoch 13/100
7/7 [==============================] - 0s 25ms/step - loss: 0.2975 - accuracy: 0.8
873 - val_loss: 0.1690 - val_accuracy: 0.9441
Epoch 14/100
7/7 [==============================] - 0s 18ms/step - loss: 0.2657 - accuracy: 0.8
944 - val_loss: 0.1547 - val_accuracy: 0.9510
Epoch 15/100
7/7 [==============================] - 0s 17ms/step - loss: 0.2587 - accuracy: 0.9
038 - val_loss: 0.1445 - val_accuracy: 0.9510
Epoch 16/100
7/7 [==============================] - 0s 16ms/step - loss: 0.2187 - accuracy: 0.9
225 - val_loss: 0.1345 - val_accuracy: 0.9510
Epoch 17/100
7/7 [==============================] - 0s 25ms/step - loss: 0.2115 - accuracy: 0.9
249 - val_loss: 0.1326 - val_accuracy: 0.9650
Epoch 18/100
7/7 [==============================] - 0s 18ms/step - loss: 0.2195 - accuracy: 0.9
108 - val_loss: 0.1213 - val_accuracy: 0.9580
```



```
7/7 [==============================] - 0s 21ms/step - loss: 0.2042 - accuracy: 0.9
```

```
296 - val_loss: 0.1130 - val_accuracy: 0.9580
Epoch 20/100
7/7 [==============================] - 0s 22ms/step - loss: 0.1923 - accuracy: 0.9
225 - val_loss: 0.1140 - val_accuracy: 0.9580
Epoch 21/100
7/7 [==============================] - 0s 23ms/step - loss: 0.1891 - accuracy: 0.9
249 - val_loss: 0.1075 - val_accuracy: 0.9720
Epoch 22/100
7/7 [==============================] - 0s 26ms/step - loss: 0.1874 - accuracy: 0.9
343 - val_loss: 0.1001 - val_accuracy: 0.9720
Epoch 23/100
7/7 [==============================] - 0s 22ms/step - loss: 0.1704 - accuracy: 0.9
413 - val_loss: 0.0997 - val_accuracy: 0.9720
Epoch 24/100
7/7 [==============================] - 0s 21ms/step - loss: 0.1617 - accuracy: 0.9
413 - val_loss: 0.0942 - val_accuracy: 0.9720
Epoch 25/100
7/7 [==============================] - 0s 20ms/step - loss: 0.1581 - accuracy: 0.9
437 - val_loss: 0.0874 - val_accuracy: 0.9580
Epoch 26/100
7/7 [==============================] - 0s 15ms/step - loss: 0.1587 - accuracy: 0.9
366 - val_loss: 0.0883 - val_accuracy: 0.9790
Epoch 27/100
7/7 [==============================] - 0s 16ms/step - loss: 0.1375 - accuracy: 0.9
531 - val_loss: 0.0889 - val_accuracy: 0.9790
Epoch 28/100
7/7 [==============================] - 0s 19ms/step - loss: 0.1456 - accuracy: 0.9
460 - val_loss: 0.0813 - val_accuracy: 0.9790
Epoch 29/100
7/7 [==============================] - 0s 19ms/step - loss: 0.1420 - accuracy: 0.9
366 - val_loss: 0.0761 - val_accuracy: 0.9650
Epoch 30/100
7/7 [==============================] - 0s 18ms/step - loss: 0.1402 - accuracy: 0.9
460 - val_loss: 0.0886 - val_accuracy: 0.9790
Epoch 31/100
7/7 [==============================] - 0s 20ms/step - loss: 0.1225 - accuracy: 0.9
601 - val_loss: 0.0750 - val_accuracy: 0.9790
Epoch 32/100
7/7 [==============================] - 0s 25ms/step - loss: 0.1443 - accuracy: 0.9
413 - val_loss: 0.0767 - val_accuracy: 0.9790
Epoch 33/100
7/7 [==============================] - 0s 21ms/step - loss: 0.1262 - accuracy: 0.9
577 - val_loss: 0.0695 - val_accuracy: 0.9720
Epoch 34/100
7/7 [==============================] - 0s 26ms/step - loss: 0.1152 - accuracy: 0.9
624 - val_loss: 0.0704 - val_accuracy: 0.9790
Epoch 35/100
7/7 [==============================] - 0s 29ms/step - loss: 0.1179 - accuracy: 0.9
507 - val_loss: 0.0699 - val_accuracy: 0.9790
Epoch 36/100
7/7 [==============================] - 0s 30ms/step - loss: 0.1131 - accuracy: 0.9
601 - val_loss: 0.0662 - val_accuracy: 0.9790
Epoch 37/100
7/7 [==============================] - 0s 21ms/step - loss: 0.1140 - accuracy: 0.9
648 - val_loss: 0.0632 - val_accuracy: 0.9790
Epoch 38/100
7/7 [==============================] - 0s 21ms/step - loss: 0.1045 - accuracy: 0.9
671 - val_loss: 0.0699 - val_accuracy: 0.9790
Epoch 39/100
7/7 [==============================] - 0s 26ms/step - loss: 0.1189 - accuracy: 0.9
577 - val_loss: 0.0678 - val_accuracy: 0.9790
Epoch 40/100
7/7 [==============================] - 0s 22ms/step - loss: 0.1033 - accuracy: 0.9
601 - val_loss: 0.0615 - val_accuracy: 0.9790
```

```
Epoch 41/100
7/7 [==============================] - 0s 21ms/step - loss: 0.1067 - accuracy: 0.9
648 - val_loss: 0.0588 - val_accuracy: 0.9790
Epoch 42/100
7/7 [==============================] - 0s 21ms/step - loss: 0.1028 - accuracy: 0.9
624 - val_loss: 0.0752 - val_accuracy: 0.9720
Epoch 43/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0913 - accuracy: 0.9
695 - val_loss: 0.0585 - val_accuracy: 0.9790
Epoch 44/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0744 - accuracy: 0.9
789 - val_loss: 0.0559 - val_accuracy: 0.9790
Epoch 45/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0945 - accuracy: 0.9
742 - val_loss: 0.0596 - val_accuracy: 0.9790
Epoch 46/100
7/7 [==============================] - 0s 18ms/step - loss: 0.1087 - accuracy: 0.9
624 - val_loss: 0.0639 - val_accuracy: 0.9790
Epoch 47/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0750 - accuracy: 0.9
812 - val_loss: 0.0549 - val_accuracy: 0.9790
Epoch 48/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0904 - accuracy: 0.9
624 - val_loss: 0.0542 - val_accuracy: 0.9790
Epoch 49/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0960 - accuracy: 0.9
601 - val_loss: 0.0554 - val_accuracy: 0.9790
Epoch 50/100
7/7 [==============================] - 0s 18ms/step - loss: 0.0856 - accuracy: 0.9
671 - val_loss: 0.0528 - val_accuracy: 0.9790
Epoch 51/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0778 - accuracy: 0.9
695 - val_loss: 0.0561 - val_accuracy: 0.9790
Epoch 52/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0988 - accuracy: 0.9
624 - val_loss: 0.0577 - val_accuracy: 0.9790
Epoch 53/100
7/7 [==============================] - 0s 27ms/step - loss: 0.0864 - accuracy: 0.9
601 - val_loss: 0.0518 - val_accuracy: 0.9790
Epoch 54/100
7/7 [==============================] - 0s 18ms/step - loss: 0.0754 - accuracy: 0.9
789 - val_loss: 0.0541 - val_accuracy: 0.9790
Epoch 55/100
7/7 [==============================] - 0s 23ms/step - loss: 0.0857 - accuracy: 0.9
742 - val_loss: 0.0570 - val_accuracy: 0.9790
Epoch 56/100
7/7 [==============================] - 0s 25ms/step - loss: 0.0765 - accuracy: 0.9
718 - val_loss: 0.0515 - val_accuracy: 0.9720
Epoch 57/100
7/7 [==============================] - 0s 18ms/step - loss: 0.0792 - accuracy: 0.9
695 - val_loss: 0.0514 - val_accuracy: 0.9720
Epoch 58/100
7/7 [==============================] - 0s 22ms/step - loss: 0.0697 - accuracy: 0.9
718 - val_loss: 0.0686 - val_accuracy: 0.9720
Epoch 59/100
7/7 [==============================] - 0s 16ms/step - loss: 0.0745 - accuracy: 0.9
812 - val_loss: 0.0564 - val_accuracy: 0.9790
Epoch 60/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0841 - accuracy: 0.9
695 - val_loss: 0.0522 - val_accuracy: 0.9720
Epoch 61/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0711 - accuracy: 0.9
                                    y: 0.9790
Epoch 62/100
```

```
7/7 [==============================] - 0s 19ms/step - loss: 0.0759 - accuracy: 0.9
742 - val_loss: 0.0531 - val_accuracy: 0.9790
Epoch 63/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0652 - accuracy: 0.9
718 - val_loss: 0.0515 - val_accuracy: 0.9720
Epoch 64/100
7/7 [==============================] - 0s 22ms/step - loss: 0.0726 - accuracy: 0.9
765 - val_loss: 0.0546 - val_accuracy: 0.9790
Epoch 65/100
7/7 [==============================] - 0s 25ms/step - loss: 0.0754 - accuracy: 0.9
718 - val_loss: 0.0524 - val_accuracy: 0.9790
Epoch 66/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0791 - accuracy: 0.9
718 - val_loss: 0.0602 - val_accuracy: 0.9720
Epoch 67/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0821 - accuracy: 0.9
742 - val_loss: 0.0490 - val_accuracy: 0.9790
Epoch 68/100
7/7 [==============================] - 0s 26ms/step - loss: 0.0738 - accuracy: 0.9
765 - val_loss: 0.0523 - val_accuracy: 0.9790
Epoch 69/100
7/7 [==============================] - 0s 22ms/step - loss: 0.0753 - accuracy: 0.9
765 - val_loss: 0.0526 - val_accuracy: 0.9790
Epoch 70/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0747 - accuracy: 0.9
765 - val_loss: 0.0495 - val_accuracy: 0.9860
Epoch 71/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0530 - accuracy: 0.9
836 - val_loss: 0.0502 - val_accuracy: 0.9790
Epoch 72/100
7/7 [==============================] - 0s 21ms/step - loss: 0.0793 - accuracy: 0.9
789 - val_loss: 0.0505 - val_accuracy: 0.9790
Epoch 73/100
7/7 [==============================] - 0s 22ms/step - loss: 0.0766 - accuracy: 0.9
742 - val_loss: 0.0594 - val_accuracy: 0.9720
Epoch 74/100
7/7 [==============================] - 0s 23ms/step - loss: 0.0650 - accuracy: 0.9
789 - val_loss: 0.0517 - val_accuracy: 0.9790
Epoch 75/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0617 - accuracy: 0.9
836 - val_loss: 0.0496 - val_accuracy: 0.9790
Epoch 76/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0568 - accuracy: 0.9
883 - val_loss: 0.0528 - val_accuracy: 0.9790
Epoch 77/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0634 - accuracy: 0.9
765 - val_loss: 0.0591 - val_accuracy: 0.9790
Epoch 78/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0713 - accuracy: 0.9
695 - val_loss: 0.0484 - val_accuracy: 0.9790
Epoch 79/100
7/7 [==============================] - 0s 13ms/step - loss: 0.0646 - accuracy: 0.9
812 - val_loss: 0.0546 - val_accuracy: 0.9790
Epoch 80/100
7/7 [==============================] - 0s 16ms/step - loss: 0.0887 - accuracy: 0.9
648 - val_loss: 0.0531 - val_accuracy: 0.9790
Epoch 81/100
7/7 [==============================] - 0s 31ms/step - loss: 0.0637 - accuracy: 0.9
812 - val_loss: 0.0489 - val_accuracy: 0.9790
Epoch 82/100
7/7 [==============================] - 0s 25ms/step - loss: 0.0544 - accuracy: 0.9
812 - val_loss: 0.0480 - val_accuracy: 0.9860
```



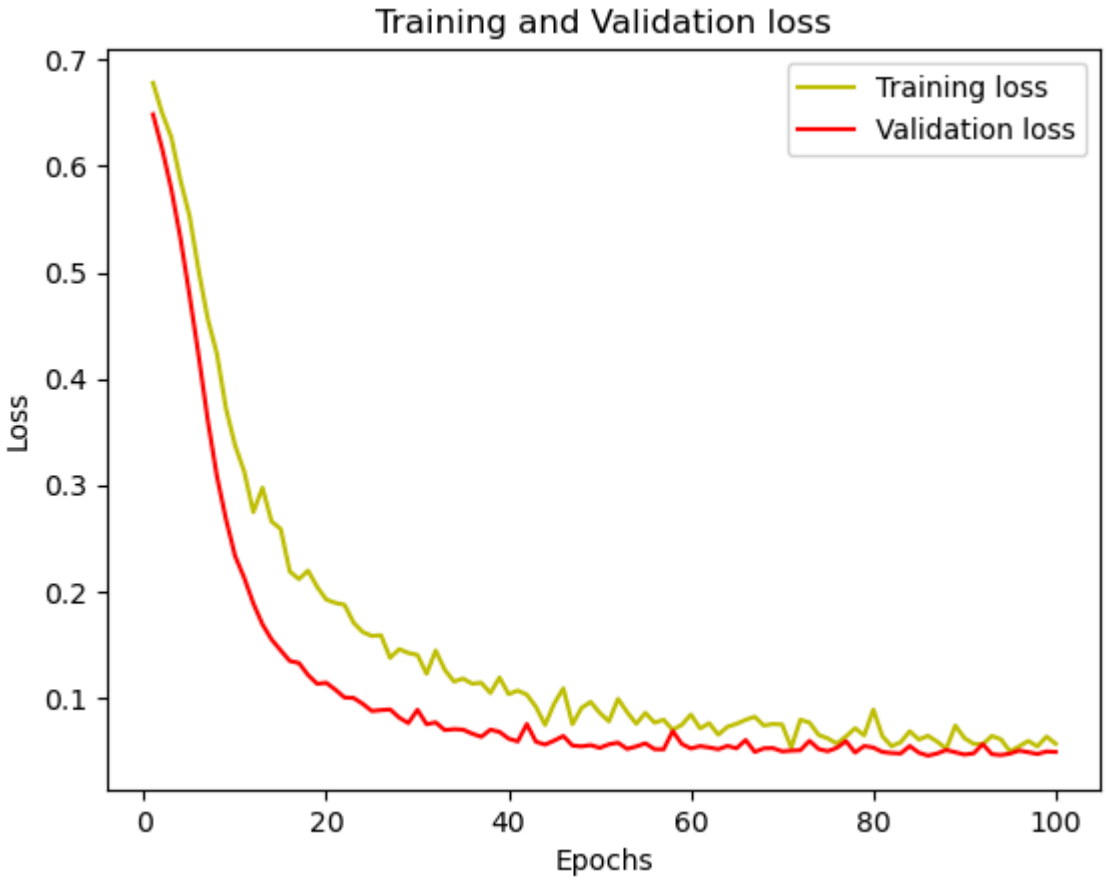
```
7/7 [==============================] - 0s 20ms/step - loss: 0.0581 - accuracy: 0.9
```

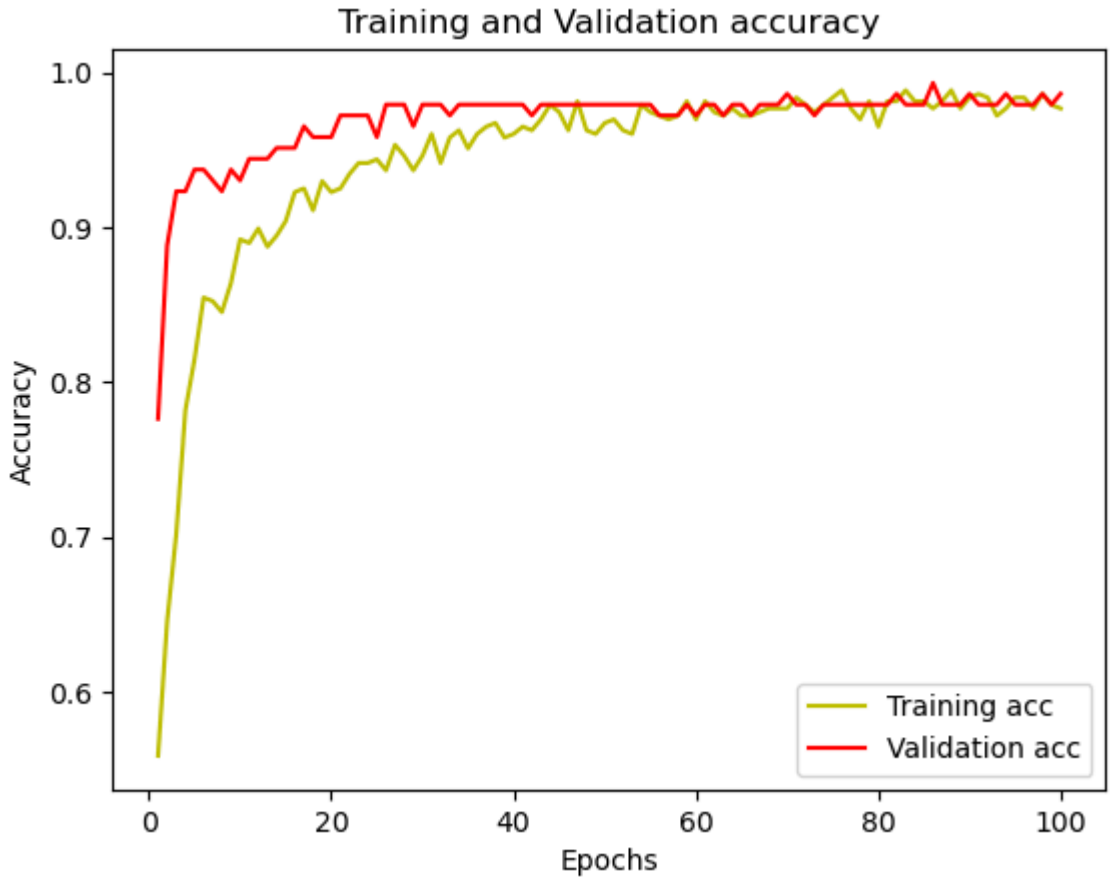
```
883 - val_loss: 0.0475 - val_accuracy: 0.9790
Epoch 84/100
7/7 [==============================] - 0s 27ms/step - loss: 0.0685 - accuracy: 0.9
812 - val_loss: 0.0545 - val_accuracy: 0.9790
Epoch 85/100
7/7 [==============================] - 0s 21ms/step - loss: 0.0605 - accuracy: 0.9
812 - val_loss: 0.0482 - val_accuracy: 0.9790
Epoch 86/100
7/7 [==============================] - 0s 21ms/step - loss: 0.0641 - accuracy: 0.9
765 - val_loss: 0.0454 - val_accuracy: 0.9930
Epoch 87/100
7/7 [==============================] - 0s 18ms/step - loss: 0.0587 - accuracy: 0.9
812 - val_loss: 0.0475 - val_accuracy: 0.9790
Epoch 88/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0520 - accuracy: 0.9
883 - val_loss: 0.0508 - val_accuracy: 0.9790
Epoch 89/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0737 - accuracy: 0.9
765 - val_loss: 0.0487 - val_accuracy: 0.9790
Epoch 90/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0619 - accuracy: 0.9
836 - val_loss: 0.0465 - val_accuracy: 0.9860
Epoch 91/100
7/7 [==============================] - 0s 22ms/step - loss: 0.0566 - accuracy: 0.9
859 - val_loss: 0.0476 - val_accuracy: 0.9790
Epoch 92/100
7/7 [==============================] - 0s 24ms/step - loss: 0.0562 - accuracy: 0.9
836 - val_loss: 0.0563 - val_accuracy: 0.9790
Epoch 93/100
7/7 [==============================] - 0s 18ms/step - loss: 0.0640 - accuracy: 0.9
718 - val_loss: 0.0469 - val_accuracy: 0.9790
Epoch 94/100
7/7 [==============================] - 0s 18ms/step - loss: 0.0608 - accuracy: 0.9
765 - val_loss: 0.0460 - val_accuracy: 0.9860
Epoch 95/100
7/7 [==============================] - 0s 21ms/step - loss: 0.0494 - accuracy: 0.9
836 - val_loss: 0.0474 - val_accuracy: 0.9790
Epoch 96/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0542 - accuracy: 0.9
836 - val_loss: 0.0503 - val_accuracy: 0.9790
Epoch 97/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0593 - accuracy: 0.9
765 - val_loss: 0.0487 - val_accuracy: 0.9790
Epoch 98/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0542 - accuracy: 0.9
859 - val_loss: 0.0470 - val_accuracy: 0.9860
Epoch 99/100
7/7 [==============================] - 0s 19ms/step - loss: 0.0634 - accuracy: 0.9
789 - val_loss: 0.0494 - val_accuracy: 0.9790
Epoch 100/100
7/7 [==============================] - 0s 21ms/step - loss: 0.0566 - accuracy: 0.9
765 - val_loss: 0.0489 - val_accuracy: 0.9860
```

In [28]:

```
# plot the training and validation accuracy and loss at each epochs:
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1,len(loss)+1)
plt.plot(epochs,loss,'y',label = 'Training loss')
plt.plot(epochs,val_loss,'r',label = 'Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs,acc,'y',label = 'Training acc')
plt.plot(epochs,val_acc,'r',label = 'Validation acc')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





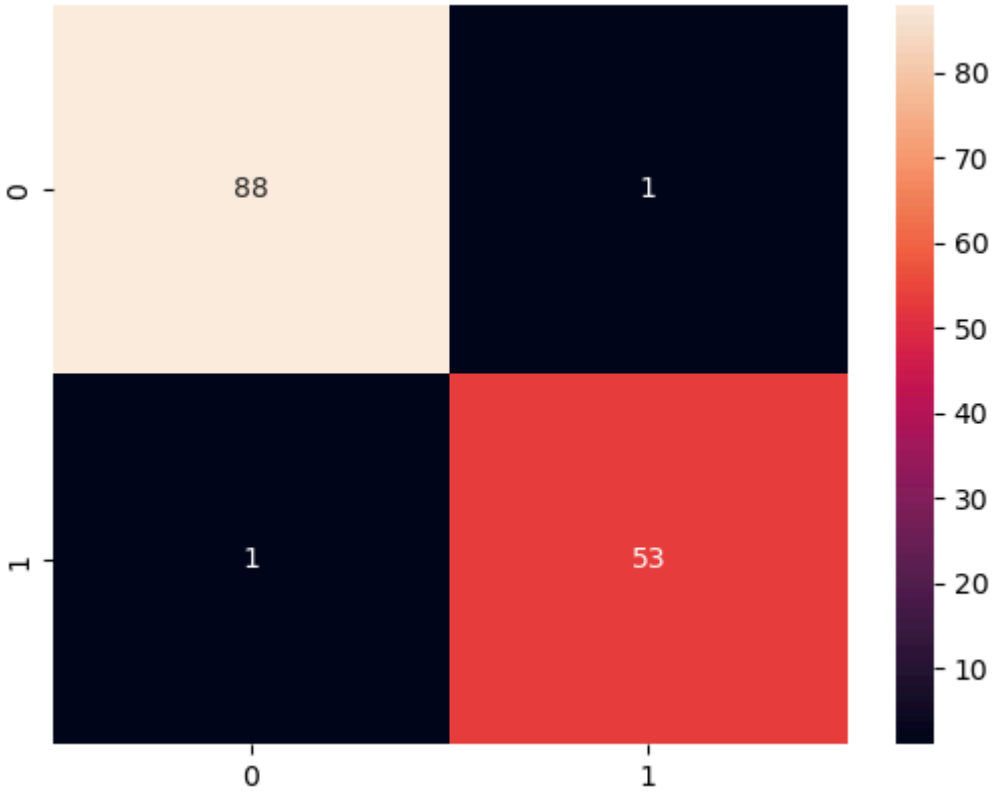
In [29]:
```
# Predicting the Test set results:
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5)

# Making the Confusion Matrix:
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)

sns.heatmap(cm, annot = True)
```

```
5/5 [==============================] - 0s 5ms/step
```

Out[29]:
```
<Axes: >
```



In [ ]: