BREAST CANCER CLASSIFICATION USING NEURAL NETWORKS

In [46]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [47]:
```
# Load the dataset
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data.csv to data (2).csv
```

In [48]:
```
df = pd.read_csv('data.csv')
df.head(10)
```

Out[48]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	tex
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	
5	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	...	
6	844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	...	
7	84458202	M	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	...	
8	844981	M	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	...	
9	84501001	M	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	...	

10 rows × 33 columns

In [49]:
```
# count the number of rows and columns in dataset:
df.shape
```

Out[49]: (569, 33)

In [50]:
```
# count the number of empty values in each columns:
df.isna().sum()
```

Out[50]:
```
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
```

dtype: int64

In [51]:
```
# drop the columns with all the missing values:
df = df.dropna(axis = 1)
```

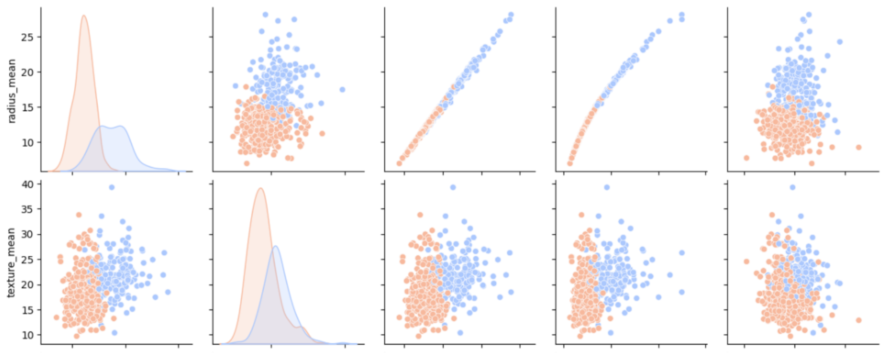
In [52]:
```
df.shape
```

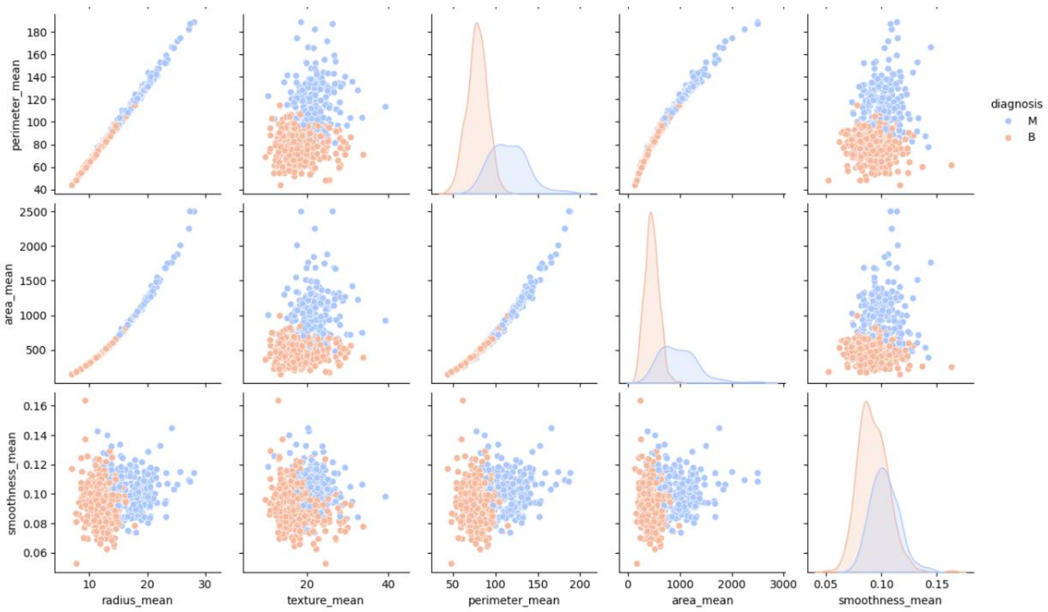
Out[52]:
```
(569, 32)
```

Let's create a pairplot that will show us the complete relationship between radius mean, texture mean, perimeter mean, area mean and smoothness mean on the basis of diagnosis type.

In [53]:
```
sns.pairplot(df,hue = 'diagnosis', palette= 'coolwarm', vars = ['radius_mean', 'texture_mean', 'perimeter_mean','area_mean','smoo
```

Out[53]:
```
<seaborn.axisgrid.PairGrid at 0x7a804cf4dde0>
```





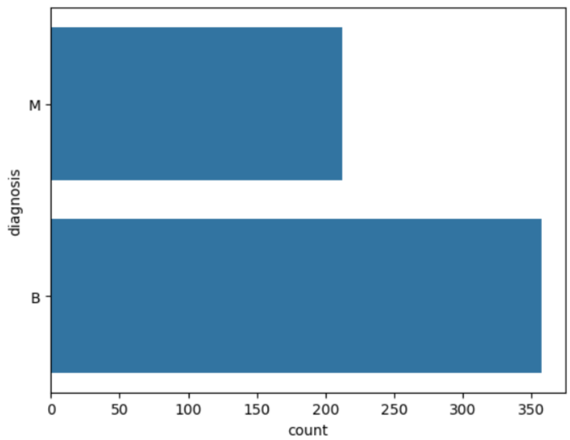
In [54]:
```python
# count the number of empty values in each columns:
df.isna().sum()

# drop the columns with all the missing values:
df = df.dropna(axis = 1)

df.shape

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

Out[54]:
```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

Now we will visualize the diagnosis column in our dataset to see how many malignant and benign are present.

In [55]:
```python
# visualize the count:
sns.countplot(df['diagnosis'], label = 'count')
```

Out[55]: <Axes: xlabel='count', ylabel='diagnosis'>



In [56]:
```
# look at the data types to see which columns need to be encoded:
df.dtypes
```

Out[56]:
```
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
dtype: object
```

In [57]:
```
# Rename the diagnosis 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 [58]:
```
# define the dependent variable that need to predict(label)
y = df['label'].values
print(np.unique(y))
```

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

In [59]:
```
# 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 [60]:
```python
# 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)

print(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]]
```

Splitting Our data:

In [61]:
```python
# Split data into training and testing data to verify accuracy after fitting the model
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_state=42)
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)
```

Creating Model, Compile and fit ml model to our training data:

In [62]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2

# Define the upgraded model
model = Sequential()

# Input layer with L2 regularization
model.add(Dense(128, input_dim=30, kernel_regularizer=l2(0.001), activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Additional hidden layers with L2 regularization and dropout
model.add(Dense(256, kernel_regularizer=l2(0.001), activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(128, kernel_regularizer=l2(0.001), activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(64, kernel_regularizer=l2(0.001), activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Output layer with sigmoid activation
model.add(Dense(1, activation='sigmoid'))

# Compile the model with Adam optimizer and binary crossentropy loss
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [63]:
```
model.summary()
```

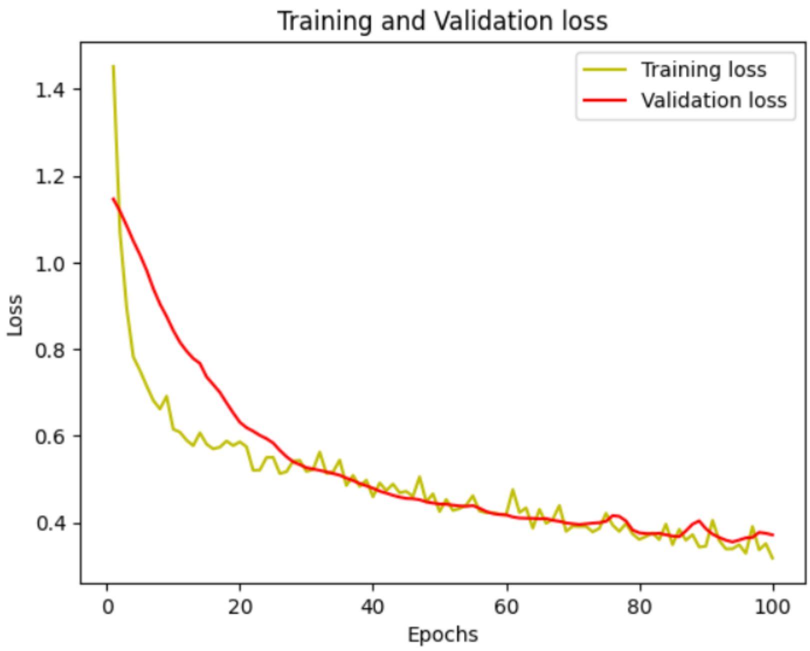
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_10 (Dense)            (None, 128)               3968      
                                                                 
 batch_normalization_8 (Bat  (None, 128)               512       
 chNormalization)                                                
                                                                 
 dropout_8 (Dropout)         (None, 128)               0         
                                                                 
 dense_11 (Dense)            (None, 256)               33024     
                                                                 
 batch_normalization_9 (Bat  (None, 256)               1024      
 chNormalization)                                                
                                                                 
 dropout_9 (Dropout)         (None, 256)               0         
                                                                 
 dense_12 (Dense)            (None, 128)               32896     
                                                                 
 batch_normalization_10 (Ba  (None, 128)               512       
 tchNormalization)                                               
                                                                 
 dropout_10 (Dropout)        (None, 128)               0         
                                                                 
 dense_13 (Dense)            (None, 64)                8256      
                                                                 
 batch_normalization_11 (Ba  (None, 64)                256       
 tchNormalization)                                               
                                                                 
 dropout_11 (Dropout)        (None, 64)                0         
                                                                 
 dense_14 (Dense)            (None, 1)                 65        
                                                                 
=================================================================
Total params: 80513 (314.50 KB)
Trainable params: 79361 (310.00 KB)
Non-trainable params: 1152 (4.50 KB)
_________________________________________________________________
```

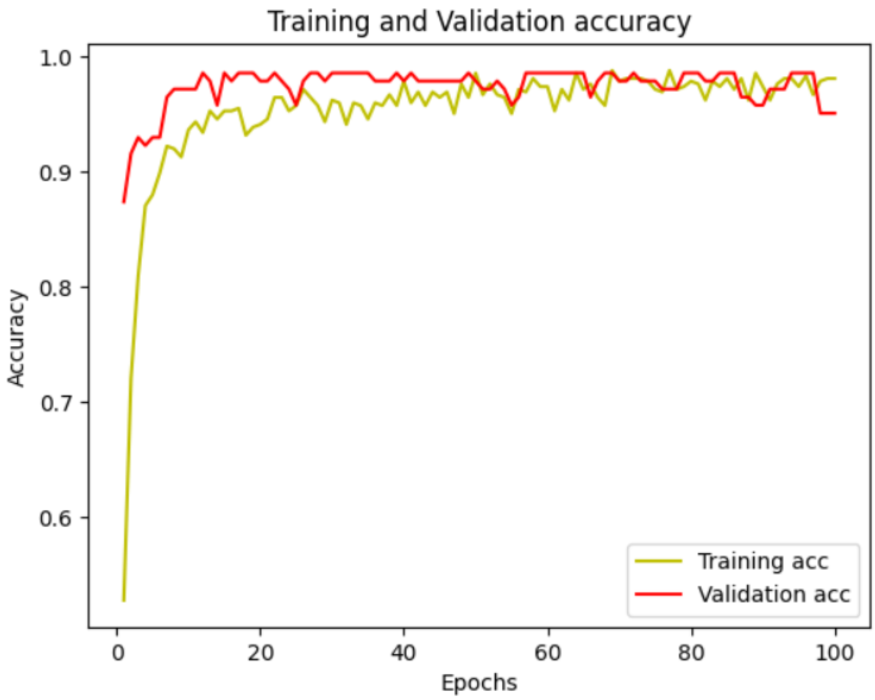
In [64]:
```
# fit with no early stopping or other callbacks:
history = model.fit(x_train,y_train,verbose = 1,epochs = 100, batch_size = 64,validation_data = (x_test,y_test))
```

```
Epoch 1/100
7/7 [==============================] - 3s 75ms/step - loss: 1.4523 - accuracy: 0.5282 - val_loss: 1.1462 - val_accuracy: 0.87
41
Epoch 2/100
7/7 [==============================] - 0s 14ms/step - loss: 1.0686 - accuracy: 0.7207 - val_loss: 1.1180 - val_accuracy: 0.91
61
Epoch 3/100
7/7 [==============================] - 0s 16ms/step - loss: 0.8950 - accuracy: 0.8099 - val_loss: 1.0847 - val_accuracy: 0.93
01
Epoch 4/100
7/7 [==============================] - 0s 15ms/step - loss: 0.7819 - accuracy: 0.8709 - val_loss: 1.0494 - val_accuracy: 0.92
31
Epoch 5/100
7/7 [==============================] - 0s 16ms/step - loss: 0.7506 - accuracy: 0.8803 - val_loss: 1.0175 - val_accuracy: 0.93
01
Epoch 6/100
7/7 [==============================] - 0s 15ms/step - loss: 0.7152 - accuracy: 0.8991 - val_loss: 0.9819 - val_accuracy: 0.93
01
Epoch 7/100
```

Visualizing our training accuracy and validation accuracy:

In [65]:
```
# 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.legend()
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()
```





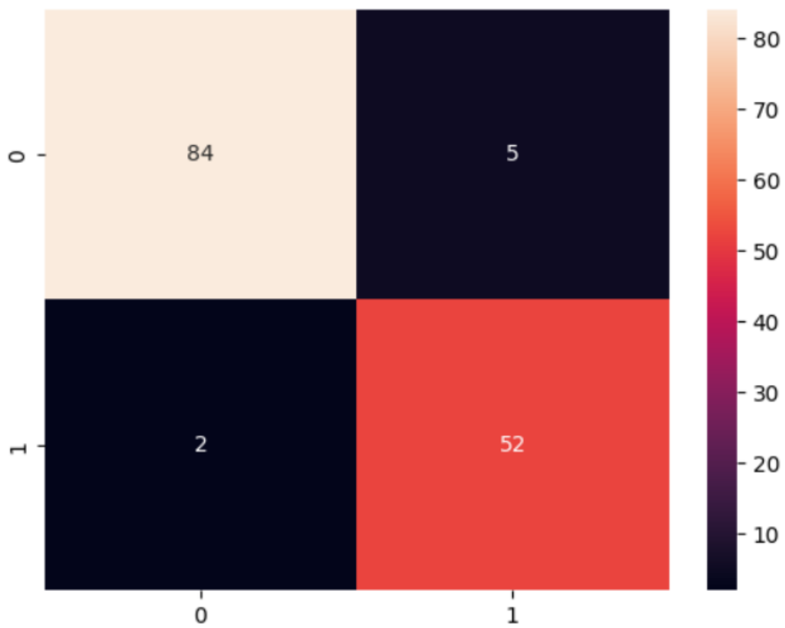
Prediction and Visualizing our model accuracy on test data: ¶

In [66]:

```
# 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 2ms/step
```

Out[66]: <Axes: >



In [67]:

```python
# Define a function to make predictions for a given patient's details
def predict_breast_cancer(model, scaler, patient_details):
    # Preprocess patient details (assuming patient_details is a list or array)
    patient_data = np.array(patient_details).reshape(1, -1)  # Reshape to 2D array

    # Scale/normalize patient data using the same scaler used for training data
    patient_data_scaled = scaler.transform(patient_data)

    # Predict the outcome using the trained model
    prediction = model.predict(patient_data_scaled)

    # Convert prediction to human-readable format (1: Malignant, 0: Benign)
    if prediction > 0.5:
        return "Malignant"
    else:
        return "Benign"

patient_details = [17.99, 10.38, 122.8, 1001, 0.1184, 0.2776, 0.3001, 0.1471, 0.2419, 0.07871,
                   1.095, 0.9053, 8.589, 153.4, 0.006399, 0.04904, 0.05373, 0.01587, 0.03003,
                   0.006193, 25.38, 17.33, 184.6, 2019, 0.1622, 0.6656, 0.7119, 0.2654,
                   0.4601, 0.1189]

# Make predictions for the example patient
predicted_diagnosis = predict_breast_cancer(model, scaler, patient_details)
print("Predicted Diagnosis for the Patient:", predicted_diagnosis)
```

```
1/1 [==============================] - 0s 18ms/step
Predicted Diagnosis for the Patient: Malignant
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MinMaxScaler
was fitted with feature names
  warnings.warn(
```

In [68]:

```python
# Define a function to make predictions for a given patient's details
def predict_breast_cancer(model, scaler, patient_details):
    # Preprocess patient details (assuming patient_details is a list or array)
    patient_data = np.array(patient_details).reshape(1, -1)  # Reshape to 2D array

    # Scale/normalize patient data using the same scaler used for training data
    patient_data_scaled = scaler.transform(patient_data)

    # Predict the outcome using the trained model
    prediction = model.predict(patient_data_scaled)

    # Convert prediction to human-readable format (1: Malignant, 0: Benign)
    if prediction > 0.5:
        return "Malignant"
    else:
        return "Benign"

patient_details = [13.54, 14.36, 87.46, 566.3, 0.09779, 0.08129, 0.06664, 0.04781, 0.1885,
                   0.05766, 0.2699, 0.7886, 2.058, 23.56, 0.008462, 0.0146, 0.02387,
                   0.01315, 0.0198, 0.0023, 15.11, 19.26, 99.7, 711.2, 0.144, 0.1773,
                   0.239, 0.1288, 0.2977, 0.07259]
# Make predictions for the example patient
predicted_diagnosis = predict_breast_cancer(model, scaler, patient_details)
print("Predicted Diagnosis for the Patient:", predicted_diagnosis)
```

```
1/1 [==============================] - 0s 20ms/step
Predicted Diagnosis for the Patient: Benign
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MinMaxScaler
was fitted with feature names
  warnings.warn(
```