Breast Cancer Classification with a simple Neural Network (NN)

Importing the Dependencies

In [3]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Collection & Processing

In [4]:
```
df = pd.read_csv("data.csv")
df.drop("id", axis=1, inplace=True)
df['diagnosis'] = df['diagnosis'].map({'M':0, 'B':1})
```

In [5]:
```
df
```

Out[5]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoo
0	0	17.99	10.38	122.80	1001.0	
1	0	20.57	17.77	132.90	1326.0	
2	0	19.69	21.25	130.00	1203.0	
3	0	11.42	20.38	77.58	386.1	
4	0	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	
564	0	21.56	22.39	142.00	1479.0	
565	0	20.13	28.25	131.20	1261.0	
566	0	16.60	28.08	108.30	858.1	
567	0	20.60	29.33	140.10	1265.0	
568	1	7.76	24.54	47.92	181.0	

569 rows × 31 columns

In [6]:
```
# print last 5 rows of the dataframe
df.tail()
```

Out[6]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoo
564	0	21.56	22.39	142.00	1479.0	
565	0	20.13	28.25	131.20	1261.0	
566	0	16.60	28.08	108.30	858.1	
567	0	20.60	29.33	140.10	1265.0	
568	1	7.76	24.54	47.92	181.0	

5 rows × 31 columns

In [7]:
```
# number of rows and columns in the dataset
df.shape
```

Out[7]: (569, 31)

In [8]:
```
# getting some information about the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   diagnosis                569 non-null    int64  
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

In [9]:

```
# checking for missing values
df.isnull().sum()
```

Out[9]:

	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

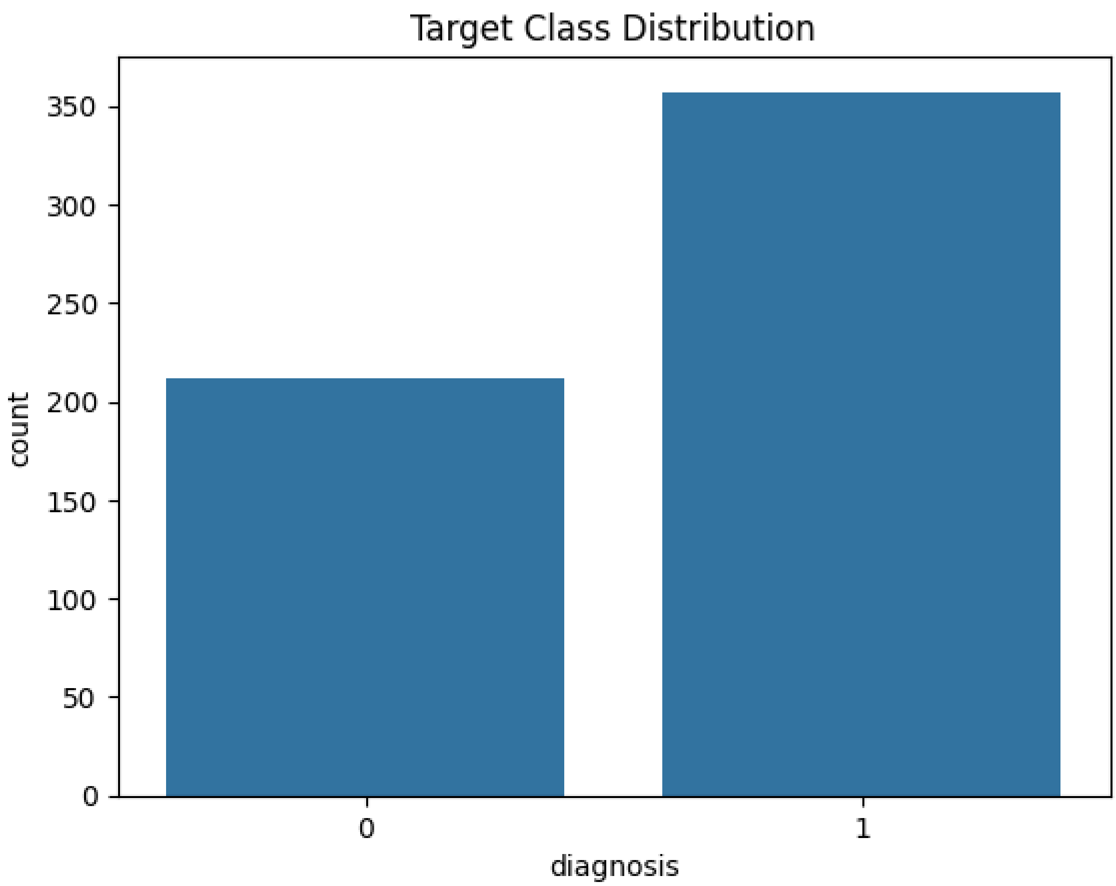
In [10]:
```python
# statistical measures about the data
df.describe()
```

Out[10]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	s
count	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	0.627417	14.127292	19.289649	91.969033	654.889104	
std	0.483918	3.524049	4.301036	24.298981	351.914129	
min	0.000000	6.981000	9.710000	43.790000	143.500000	
25%	0.000000	11.700000	16.170000	75.170000	420.300000	
50%	1.000000	13.370000	18.840000	86.240000	551.100000	
75%	1.000000	15.780000	21.800000	104.100000	782.700000	
max	1.000000	28.110000	39.280000	188.500000	2501.000000	

8 rows × 31 columns

In [11]:
```python
# Class distribution
sns.countplot(x='diagnosis', data=df)
plt.title("Target Class Distribution")
plt.show()
```



In [12]:
```
# checking the distribution of Target Varibale
df['diagnosis'].value_counts()
```

Out[12]:

diagnosis	count
1	357
0	212

dtype: int64

1 --> Benign

0 --> Malignant

In [13]:
```
df.groupby('diagnosis').mean()
```

Out[13]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
diagnosis					
0	17.462830	21.604906	115.365377	978.376415	0
1	12.146524	17.914762	78.075406	462.790196	0

2 rows × 30 columns

Separating the features and target

In [14]:

```
X = df.drop(columns='diagnosis', axis=1)
Y = df['diagnosis']
```

In [15]:

```
print(X)
```

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

     compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
..                ...             ...                  ...            ...
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     fractal_dimension_mean  ...  radius_worst  texture_worst  \
0                   0.07871  ...        25.380          17.33
1                   0.05667  ...        24.990          23.41
2                   0.05999  ...        23.570          25.53
3                   0.09744  ...        14.910          26.50
4                   0.05883  ...        22.540          16.67
..                      ...  ...           ...            ...
564                 0.05623  ...        25.450          26.40
565                 0.05533  ...        23.690          38.25
566                 0.05648  ...        18.980          34.12
567                 0.07016  ...        25.740          39.42
568                 0.05884  ...         9.456          30.37

     perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
0             184.60      2019.0           0.16220            0.66560
1             158.80      1956.0           0.12380            0.18660
2             152.50      1709.0           0.14440            0.42450
3              98.87       567.7           0.20980            0.86630
4             152.20      1575.0           0.13740            0.20500
..               ...         ...               ...                ...
564           166.10      2027.0           0.14100            0.21130
565           155.00      1731.0           0.11660            0.19220
566           126.70      1124.0           0.11390            0.30940
567           184.60      1821.0           0.16500            0.86810
568            59.16       268.6           0.08996            0.06444

     concavity_worst  concave points_worst  symmetry_worst  \
0             0.7119                0.2654          0.4601
```

```
1             0.2416              0.1860        0.2750
2             0.4504              0.2430        0.3613
3             0.6869              0.2575        0.6638
4             0.4000              0.1625        0.2364
..               ...                 ...           ...
564           0.4107              0.2216        0.2060
565           0.3215              0.1628        0.2572
566           0.3403              0.1418        0.2218
567           0.9387              0.2650        0.4087
568           0.0000              0.0000        0.2871

     fractal_dimension_worst
0                    0.11890
1                    0.08902
2                    0.08758
3                    0.17300
4                    0.07678
..                       ...
564                  0.07115
565                  0.06637
566                  0.07820
567                  0.12400
568                  0.07039

[569 rows x 30 columns]
```

In [16]:
```
print(Y)
```

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: diagnosis, Length: 569, dtype: int64
```

Splitting the data into training data & Testing data

In [17]:
```
# ◈ Splitting the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, rando
```

In [18]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(569, 30) (455, 30) (114, 30)
```

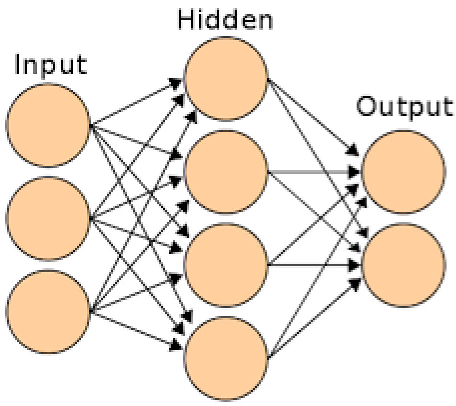
Standardize the data

In [19]:
```
scaler = StandardScaler()
```

```
X_train_std = scaler.fit_transform(X_train)

X_test_std = scaler.transform(X_test)
```

Building the Neural Network



In [20]:
```
tf.random.set_seed(3)
```

In [21]:
```
# setting up the layers of Neural Network

# Defining the Neural Network model
model = keras.Sequential([
    keras.layers.Dense(20, activation='relu', input_shape=(30,)),
    keras.layers.Dense(1, activation='sigmoid')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usi
ng Sequential models, prefer using an `Input(shape)` object as the first layer
in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [22]:
```
# compiling the Neural Network

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [23]:
```
# training the Meural Network

# ⛶ Training the model
history = model.fit(X_train_std, Y_train, validation_split=0.1, epochs=10)
```

```
Epoch 1/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.3723 - loss: 1.0333 - va
l_accuracy: 0.4565 - val_loss: 0.7439
Epoch 2/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.4782 - loss: 0.7535 - va
l_accuracy: 0.8043 - val_loss: 0.5189
Epoch 3/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6648 - loss: 0.5571 - va
l_accuracy: 0.9783 - val_loss: 0.3784
Epoch 4/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8087 - loss: 0.4306 - va
l_accuracy: 0.9783 - val_loss: 0.2954
Epoch 5/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8535 - loss: 0.3507 - va
l_accuracy: 0.9783 - val_loss: 0.2445
Epoch 6/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8787 - loss: 0.2975 - va
l_accuracy: 0.9783 - val_loss: 0.2115
Epoch 7/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9021 - loss: 0.2597 - va
l_accuracy: 0.9783 - val_loss: 0.1885
Epoch 8/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9234 - loss: 0.2311 - va
l_accuracy: 0.9783 - val_loss: 0.1715
Epoch 9/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9282 - loss: 0.2086 - va
l_accuracy: 0.9783 - val_loss: 0.1585
Epoch 10/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9405 - loss: 0.1903 - va
l_accuracy: 0.9783 - val_loss: 0.1482
```

Visualizing accuracy and loss

In [24]:

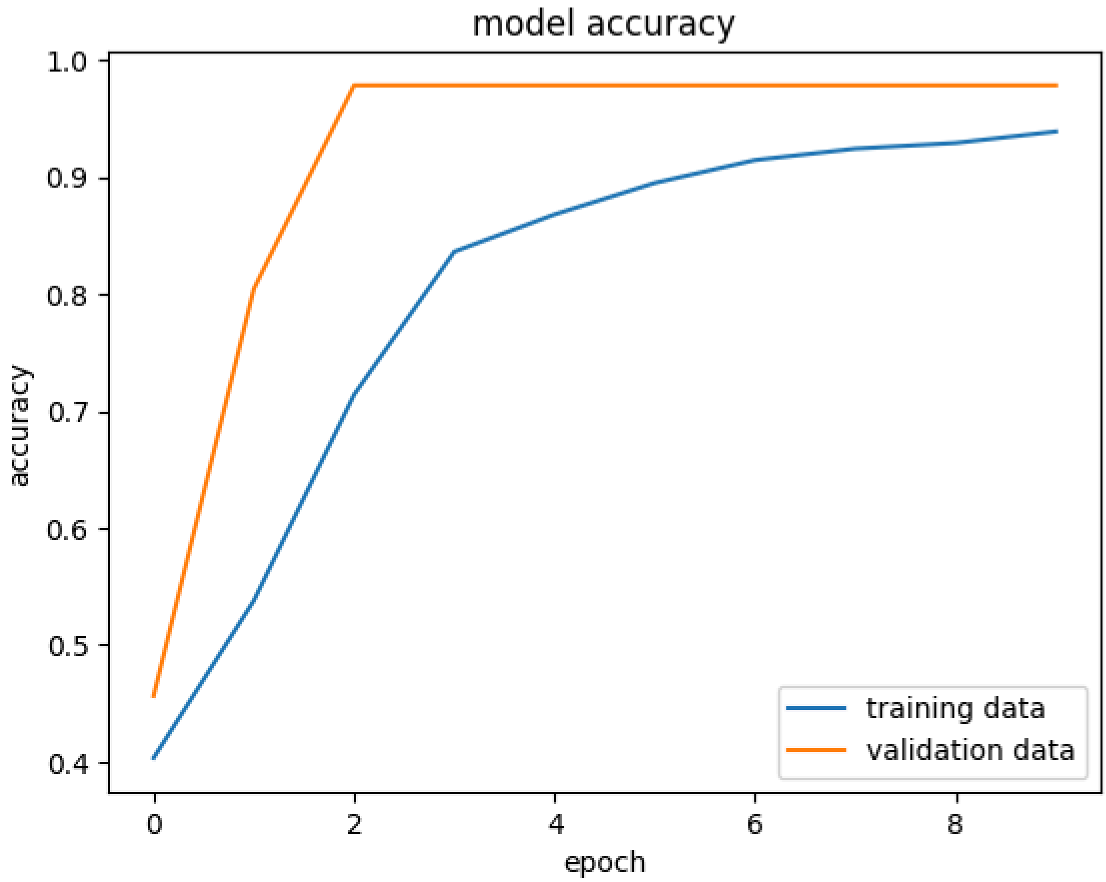
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'lower right')
```

Out[24]:

```
<matplotlib.legend.Legend at 0x7ab114d867d0>
```

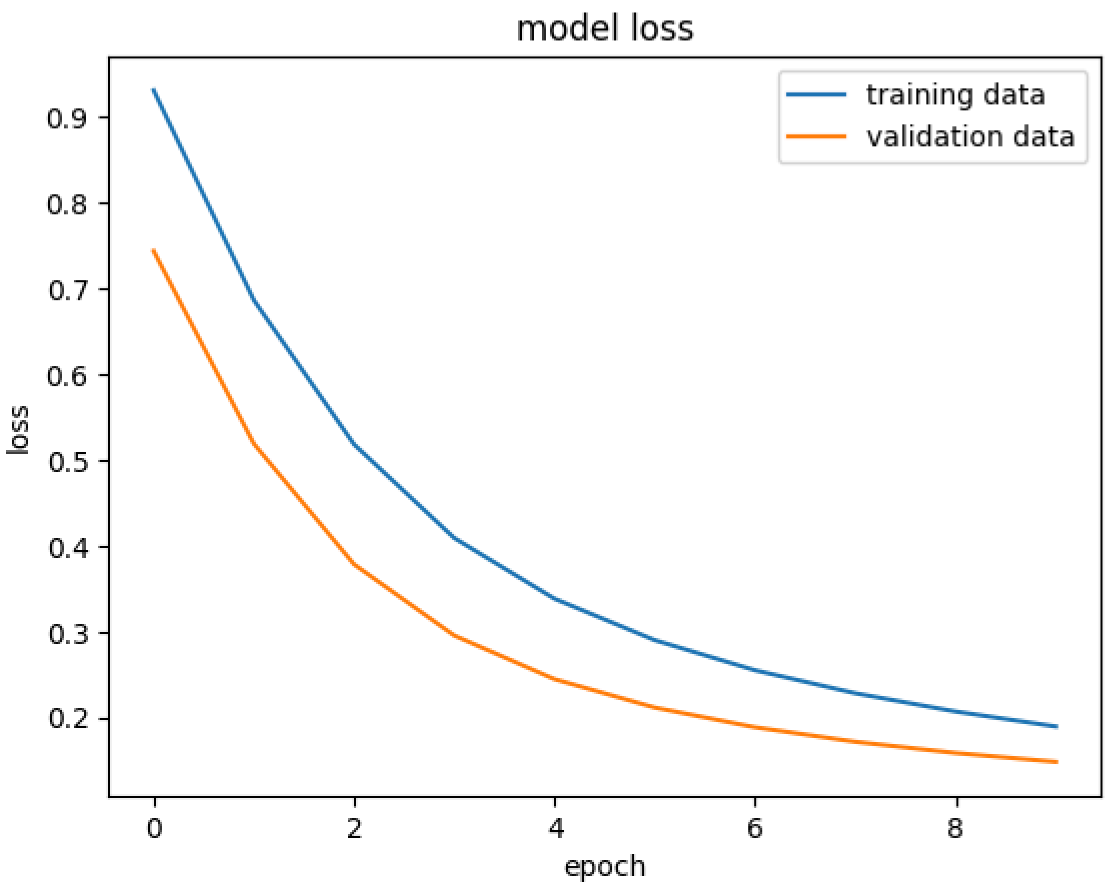


In [25]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'upper right')
```

Out[25]: <matplotlib.legend.Legend at 0x7ab176feba50>



Accuracy of the model on test data

In [26]:
```
# ◈ Evaluating the model on test data
loss, accuracy = model.evaluate(X_test_std, Y_test)
print(accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9230 - loss: 0.2199
0.9298245906829834
```

In [27]:
```
print(X_test_std.shape)
print(X_test_std[0])
```

```
(114, 30)
[-0.04462793 -1.41612656 -0.05903514 -0.16234067  2.0202457  -0.11323672
  0.18500609  0.47102419  0.63336386  0.26335737  0.53209124  2.62763999
  0.62351167  0.11405261  1.01246781  0.41126289  0.63848593  2.88971815
 -0.41675911  0.74270853 -0.32983699 -1.67435595 -0.36854552 -0.38767294
  0.32655007 -0.74858917 -0.54689089 -0.18278004 -1.23064515 -0.6268286 ]
```

In [28]:
```
Y_pred = model.predict(X_test_std)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```

In [29]:
```
print(Y_pred.shape)
print(Y_pred[0])
```

```
(114, 1)
[0.57944816]
```

In [30]:
```
print(X_test_std)
```

```
[[-0.04462793 -1.41612656 -0.05903514 ... -0.18278004 -1.23064515
  -0.6268286 ]
 [ 0.24583601 -0.06219797  0.21802678 ...  0.54129749  0.11047691
   0.0483572 ]
 [-1.26115925 -0.29051645 -1.26499659 ... -1.35138617  0.269338
  -0.28231213]
 ...
 [ 0.72709489  0.45836817  0.75277276 ...  1.46701686  1.19909344
   0.65319961]
 [ 0.25437907  1.33054477  0.15659489 ... -1.29043534 -2.22561725
  -1.59557344]
 [ 0.84100232 -0.06676434  0.8929529  ...  2.15137705  0.35629355
   0.37459546]]
```

In [31]:
```
print(Y_pred)
```

```
[[5.7944816e-01]
 [5.4799771e-01]
 [9.5382488e-01]
 [6.0756531e-05]
 [4.8952186e-01]
 [7.6010404e-03]
 [6.2022758e-01]
 [9.7304952e-01]
 [9.0360391e-01]
 [9.3224633e-01]
 [6.2372327e-01]
 [8.6918908e-01]
 [6.6956234e-01]
 [8.4866720e-01]
 [8.8146442e-01]
 [1.0968794e-02]
 [9.5920968e-01]
 [8.5751635e-01]
 [8.6840624e-01]
 [2.2778651e-02]
 [1.6535509e-01]
 [9.5268559e-01]
 [9.2469513e-01]
 [9.6149290e-01]
 [7.5835764e-01]
 [2.8978478e-02]
 [8.8868731e-01]
 [7.7837563e-01]
 [2.1731045e-02]
 [1.4245502e-02]
 [8.7887466e-01]
 [8.6644804e-01]
 [8.6391968e-01]
 [5.1455584e-04]
 [1.6218390e-02]
 [6.7806208e-01]
 [9.8855454e-01]
 [8.9672464e-01]
 [9.6932524e-01]
 [9.0333766e-01]
 [5.4680562e-04]
 [1.6645780e-01]
 [9.8668027e-01]
 [8.0902153e-01]
 [1.5716387e-01]
 [8.9677358e-01]
 [9.3963349e-01]
 [8.7662619e-01]
 [3.8213830e-04]
 [4.3045968e-02]
 [9.1671801e-01]
 [1.4556846e-01]
 [4.5559421e-01]
 [9.4883031e-01]
```

```
[9.2862117e-01]
[4.7570056e-01]
[9.1469473e-01]
[8.9343512e-01]
[1.6958913e-02]
[7.3393339e-01]
[7.0428139e-01]
[7.2126172e-02]
[9.5788360e-01]
[2.3276834e-02]
[2.6675813e-02]
[4.8119459e-01]
[1.9572754e-03]
[6.3562877e-02]
[5.7100791e-01]
[9.5968835e-02]
[1.2510759e-01]
[1.5738681e-02]
[8.3469772e-01]
[1.7510022e-01]
[9.8128551e-01]
[1.4944032e-01]
[8.9824814e-01]
[9.2390132e-01]
[5.2017534e-01]
[2.3615752e-01]
[1.0397666e-02]
[1.3247216e-01]
[3.2603938e-02]
[5.7312453e-01]
[8.5071766e-01]
[4.5505381e-01]
[8.3423293e-01]
[7.5447232e-01]
[5.6370598e-01]
[1.0403588e-02]
[8.6998773e-01]
[9.0814209e-01]
[8.2195503e-01]
[1.9545414e-02]
[1.3483056e-01]
[8.4950876e-01]
[7.0937891e-03]
[2.3816984e-02]
[9.4713300e-01]
[9.7018409e-01]
[9.7874528e-01]
[3.5689881e-01]
[2.3128826e-04]
[1.3958461e-03]
[9.0535390e-01]
[9.0754831e-01]
[8.0991483e-01]
[7.5048649e-01]
```

```
 [9.3533069e-01]
 [9.2685550e-01]
 [2.8191048e-03]
 [1.0194858e-02]
 [5.1812589e-01]
 [4.5157094e-02]]
```

model.predict() gives the prediction probability of each class for that data point

In [32]:
```python
#  argmax function

my_list = [0.25, 0.56]

index_of_max_value = np.argmax(my_list)
print(my_list)
print(index_of_max_value)
```

```
[0.25, 0.56]
1
```

In [33]:
```python
# converting the prediction probability to class labels

Y_pred_labels = [np.argmax(i) for i in Y_pred]
print(Y_pred_labels)
```

```
[np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0),
np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), n
p.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0)]
```

Building the predictive system

In [34]:
```python
def predict_cancer(input_data, model, scaler):
    """
    Predicts whether a tumor is benign or malignant based on input features.

    Parameters:
    - input_data: Tuple or list of 30 numerical values (features)
    - model: Trained Keras model
```

```
    - scaler: Trained StandardScaler instance

    Returns:
    - String indicating the result: 'Malignant' or 'Benign'
    """
    # Convert input to numpy array
    input_data_as_numpy_array = np.asarray(input_data)

    # Reshape as 2D array (single sample)
    input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)

    # Standardize input
    input_data_std = scaler.transform(input_data_reshaped)

    # Predict
    prediction = model.predict(input_data_std)

    # Binary classification threshold
    predicted_class = int(prediction[0][0] > 0.5)

    # Return result
    return 'Malignant' if predicted_class == 0 else 'Benign'
```

In [35]:

```
input_data = (11.76,21.6,74.72,427.9,0.08637,0.04966,0.01657,0.01115,0.1495,0.
              0.4062,1.21,2.635,28.47,0.005857,0.009758,0.01168,0.007445,0.024
              12.98,25.72,82.98,516.5,0.1085,0.08615,0.05523,0.03715,0.2433,0.

result = predict_cancer(input_data, model, scaler)
print("Prediction:", result)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
Prediction: Benign
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserW
arning: X does not have valid feature names, but StandardScaler was fitted with
feature names
  warnings.warn(
```