INTRODUCTION

Here our project aims to develop a neural network model that can classify the tumors as malignant or benign by using the Breast Cancer Wisconsin Dataset from `sklearn`. The goal of this project is to produce an accurate and reliable model that assists in early detection by applying machine learning to support medical diagnostics.

IMPORTING THE LIBRARIES AND NECESSARY DEPENDENCIES

In [42]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.datasets
from sklearn.model_selection import train_test_split
```

Below are the imports used:

NumPy for creating arrays and performing mathematical computations, Pandas for handling data in the form of DataFrames, and Matplotlib for visualizing results after training. Additionally, sklearn.datasets provides datasets, and train_test_split is used for splitting the data into training and testing sets.

DATA COLLECTION AND PRE PROCESSING

LOADING THE DATASET

In [43]:

```
dataset = sklearn.datasets.load_breast_cancer()
```

In [44]:

```
print(dataset)
```

```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
```

```
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]), 'frame': None, 'target_n
ames': array(['malignant', 'benign'], dtype='<U9'), 'DESCR': '.. _breast_cancer_dataset:\
n\nBreast cancer wisconsin (diagnostic) dataset\n--------------------------------------------
-----\n\n**Data Set Characteristics:**\n\n:Number of Instances: 569\n\n:Number of Attribu
tes: 30 numeric, predictive attributes and the class\n\n:Attribute Information:\n    - ra
dius (mean of distances from center to points on the perimeter)\n    - texture (standard
deviation of gray-scale values)\n    - perimeter\n    - area\n    - smoothness (local var
iation in radius lengths)\n    - compactness (perimeter^2 / area - 1.0)\n    - concavity
(severity of concave portions of the contour)\n    - concave points (number of concave po
rtions of the contour)\n    - symmetry\n    - fractal dimension ("coastline approximation
" - 1)\n\n    The mean, standard error, and "worst" or largest (mean of the three\n    wo
rst/largest values) of these features were computed for each image,\n    resulting in 30
features.  For instance, field 0 is Mean Radius, field\n    10 is Radius SE, field 20 is
Worst Radius.\n\n    - class:\n            - WDBC-Malignant\n            - WDBC-Benign\n
\n:Summary Statistics:\n\n===================================== ====== ======\n
Min    Max\n===================================== ====== ======\nradius (mean):
6.981  28.11\ntexture (mean):                       9.71   39.28\nperimeter (mean):
43.79  188.5\narea (mean):                          143.5  2501.0\nsmoothness (mean):
0.053  0.163\ncompactness (mean):                   0.019  0.345\nconcavity (mean):
0.0    0.427\nconcave points (mean):                0.0    0.201\nsymmetry (mean):
0.106  0.304\nfractal dimension (mean):             0.05   0.097\nradius (standard error)
:               0.112  2.873\ntexture (standard error):            0.36   4.885\nperimet
er (standard error):           0.757  21.98\narea (standard error):                6.802
542.2\nsmoothness (standard error):         0.002  0.031\ncompactness (standard error):
0.002  0.135\nconcavity (standard error):           0.0    0.396\nconcave points (standar
d error):      0.0    0.053\nsymmetry (standard error):            0.008  0.079\nfractal
dimension (standard error):   0.001  0.03\nradius (worst):                      7.93
36.04\ntexture (worst):                     12.02  49.54\nperimeter (worst):
50.41  251.2\narea (worst):                          185.2  4254.0\nsmoothness (worst):
0.071  0.223\ncompactness (worst):                  0.027  1.058\nconcavity (worst):
0.0    1.252\nconcave points (worst):               0.0    0.291\nsymmetry (worst):
0.156  0.664\nfractal dimension (worst):            0.055  0.208\n=======================
============== ====== ======\n\n:Missing Attribute Values: None\n\n:Class Distribution: 2
12 - Malignant, 357 - Benign\n\n:Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L
. Mangasarian\n\n:Donor: Nick Street\n\n:Date: November, 1995\n\nThis is a copy of UCI ML
Breast Cancer Wisconsin (Diagnostic) datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are com
puted from a digitized image of a fine needle\naspirate (FNA) of a breast mass.  They des
cribe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane descr
ibed above was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision
Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidwest Artificial In
telligence and Cognitive Science Society,\npp. 97-101, 1992], a classification method whi
ch uses linear\nprogramming to construct a decision tree.  Relevant features\nwere select
ed using an exhaustive search in the space of 1-4\nfeatures and 1-3 separating planes.\n\
nThe actual linear program used to obtain the separating plane\nin the 3-dimensional spac
e is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear\nProgrammin
g Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods and Software 1,
1992, 23-34].\n\nThis database is also available through the UW CS ftp server:\n\nftp ftp
.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n.. dropdown:: References\n\
n  - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction\n    for
breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on\n    Electronic Imaging
: Science and Technology, volume 1905, pages 861-870,\n    San Jose, CA, 1993.\n  - O.L.
Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and\n    prognosis via
linear programming. Operations Research, 43(4), pages 570-577,\n    July-August 1995.\n
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques\n    to di
agnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)\n    163-171.\n
', 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23'), 'filename': 'breast_c
```

```
'worst symmetry', 'worst fractal dimension'], dtype='<U23'), 'filename': 'breast_c
ancer.csv', 'data_module': 'sklearn.datasets.data'}
```

LOADING THE DATA IN DATAFRAME

In [45]:

```
finaldataset= pd.DataFrame(dataset.data , columns = dataset.feature_names )
```

In [46]:

```
finaldataset.head()
```

Out[46]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	wors textur
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.3
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.4
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.5
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.5
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.6

5 rows × 30 columns

In [47]:

```
finaldataset.tail()
```

Out[47]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	wo text
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	2
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	3
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	3
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	3
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	3

5 rows × 30 columns

ADDING THE TARGET COLUMN TO DATAFRAME

In [48]:

```
finaldataset['label']= dataset.target
```

In [49]:

```
finaldataset.head()
```

Out[49]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	w perin
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	18

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	w perim
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	15
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	15

5 rows × 31 columns

In [50]:

```
finaldataset.tail()
```

Out[50]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	pe
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	

5 rows × 31 columns

In [51]:

```
finaldataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  label                    569 non-null    int64
```

```
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

In [52]:

```
finaldataset.shape
```

Out[52]:

```
(569, 31)
```

In [53]:

```
finaldataset.describe()
```

Out[53]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	dim
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.

8 rows × 31 columns

In [54]:

```
finaldataset.isnull().sum( )
```

Out[54]:

	0
mean radius	0
mean texture	0
mean perimeter	0
mean area	0
mean smoothness	0
mean compactness	0
mean concavity	0
mean concave points	0
mean symmetry	0
mean fractal dimension	0
radius error	0
texture error	0
perimeter error	0
area error	0
smoothness error	0
compactness error	0
concavity error	0

concavity error	0
concave points error	0
symmetry error	0
fractal dimension error	0
worst radius	0
worst texture	0
worst perimeter	0
worst area	0
worst smoothness	0
worst compactness	0
worst concavity	0
worst concave points	0
worst symmetry	0
worst fractal dimension	0
label	0

dtype: int64

In [55]:

```
finaldataset['label'].value_counts()
```

Out[55]:

	count
label	
1	357
0	212

dtype: int64

in this dataset as we can see thers no imbalance on large extent so we can ignore it that we will not have to use methods like upsampeling or downsampeling.

here

1 = Benign(non-cancerous)

A benign tumor or condition is non-cancerous. It generally does not spread to other parts of the body and is not considered life-threatening. While it can cause symptoms due to its size or location, it usually does not pose a significant risk to health.

0 =Malignant(cancerous)

A malignant tumor or condition is cancerous. It has the potential to grow uncontrollably and spread to other parts of the body (a process known as metastasis). Malignant tumors can be life-threatening and often require more aggressive treatment.

In [56]:

```
finaldataset.groupby('label').mean()
```

Out[56]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	..

label	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775	0.087990	0.192909	0.062680	...
1	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058	0.025717	0.174186	0.062867	...

2 rows × 30 columns

SPLITTING THE FEATURES AND TAEGET VARIABLES

In [57]:

```
X = finaldataset.drop(columns='label',axis=1)
Y = finaldataset['label']
```

In [58]:

```
print(X)
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
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

     mean compactness  mean concavity  mean concave points  mean symmetry  \
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

     mean fractal dimension  ...  worst radius  worst texture  \
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

     worst perimeter  worst area  worst smoothness  worst compactness  \
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
```

```
     worst concavity  worst concave points  worst symmetry  \
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

     worst fractal dimension
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 [59]:

```
print(Y)
```

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

TRAIN_TEST SPLIT

In [60]:

```
X_train , X_test , Y_train , Y_test = train_test_split(X,Y,test_size=.2,random_state=42)
```

In [61]:

```
X.shape,X_train.shape,X_test.shape
```

Out[61]:

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

STANDAEDIZE THE DATA

In [62]:

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

In [63]:

```
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
```

```
X_test_std = scaler.fit_transform(X_test)
```

BUILDING A NEURAL NETWORK

In [64]:

```
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
```

TensorFlow is a deep learning library developed by Google, and it's used quite widely to build neural networks because of its extensive functionality. Keras is a wrapper for TensorFlow, often referred to as such, which makes building neural networks much easier with its very simple API. Until the appearance of TensorFlow and Keras, neural networks were considerably harder to develop.

As mentioned earlier, during the training process in neural networks, the weights and parameters get initialized randomly; hence, for each run of the model, there may be minute changes in the values of accuracy. We used `random.set_seed(3)`, which fixes this random initialization and thus always presents the same accuracy score over different runs for reproducibility.

SETTING KERAS NETWORK (CREATING LAYERS OF NN IE.INPUT LAYER,HIDDEN LAYER AND OUTPUT LAYER)

In [65]:

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(30,)),
    keras.layers.Dense(40,activation = 'relu'),
    keras.layers.Dense(2,activation = 'sigmoid')
]
)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Flatten **is used to transform the data of `X_train` and `X_test` into a uni-dimensional array. The number `30` indicates the number of input features, that is, the 30 columns in the dataset.**

In the hidden layer, the number `40` represents the number of neurons in that layer. For the output layer, 2 neurons are used because the number of neurons is equal to the number of classes in the target variable. This process is called the firing of neurons, where if one neuron outputs `1` (for example, for class 0), the other neuron will not fire (output `0`) and vice versa. This setup ensures accurate classification.

COMPLING THE NEURAL NETWORK

In [66]:

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

NOTE:

while using categorical variables in dataet like 0,1 we use 'sparse_categorical_crossentropy' and when we have one hot encoded labels we have 'categorical_crossentophy'

TRAINING THE NEURAL NETWORK

In [67]:

```
history = model.fit(X_train_std,Y_train , validation_split =.1 ,epochs=20)
```

```
Epoch 1/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.7222 - loss: 0.5641 - val_accuracy:
0.8261 - val_loss: 0.3642
Epoch 2/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8859 - loss: 0.2999 - val_accuracy:
0.8913 - val_loss: 0.2458
Epoch 3/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9237 - loss: 0.2148 - val_accuracy:
0.9348 - val_loss: 0.1891
Epoch 4/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9467 - loss: 0.1732 - val_accuracy:
0.9565 - val_loss: 0.1572
Epoch 5/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9503 - loss: 0.1478 - val_accuracy:
0.9565 - val_loss: 0.1369
Epoch 6/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9564 - loss: 0.1301 - val_accuracy:
0.9783 - val_loss: 0.1229
Epoch 7/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9564 - loss: 0.1166 - val_accuracy:
1.0000 - val_loss: 0.1123
Epoch 8/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9646 - loss: 0.1059 - val_accuracy:
1.0000 - val_loss: 0.1041
Epoch 9/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9696 - loss: 0.0972 - val_accuracy:
1.0000 - val_loss: 0.0974
Epoch 10/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9747 - loss: 0.0898 - val_accuracy:
1.0000 - val_loss: 0.0918
Epoch 11/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9795 - loss: 0.0834 - val_accuracy:
1.0000 - val_loss: 0.0871
Epoch 12/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9795 - loss: 0.0780 - val_accuracy:
1.0000 - val_loss: 0.0832
Epoch 13/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9873 - loss: 0.0733 - val_accuracy:
1.0000 - val_loss: 0.0798
Epoch 14/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9873 - loss: 0.0690 - val_accuracy:
1.0000 - val_loss: 0.0768
Epoch 15/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9873 - loss: 0.0653 - val_accuracy:
1.0000 - val_loss: 0.0743
Epoch 16/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9873 - loss: 0.0619 - val_accuracy:
1.0000 - val_loss: 0.0721
Epoch 17/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9873 - loss: 0.0588 - val_accuracy:
1.0000 - val_loss: 0.0701
Epoch 18/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9895 - loss: 0.0561 - val_accuracy:
1.0000 - val_loss: 0.0683
Epoch 19/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9869 - loss: 0.0536 - val_accuracy:
1.0000 - val_loss: 0.0667
Epoch 20/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9869 - loss: 0.0512 - val_accuracy:
1.0000 - val_loss: 0.0653
```

We can observe in the training process that when **loss** decreases, the **accuracy** score increases. It shows that as the model is learning, it is getting better in classifying the data, which is reflected by the decrease in loss and an increase in accuracy. Moreover, the **validation accuracy** is quite high, which shows that the model performs well on unseen data.

However, in the above example, it is necessary to **standardize** the data before training by using a technique such as **StandardScaler**. Standardizing the data ensures that all features have a similar scale, which helps the model converge faster and improves overall performance. This is especially important for neural networks, where feature scaling can significantly impact training efficiency and model accuracy.

visualizing accuracy and loss

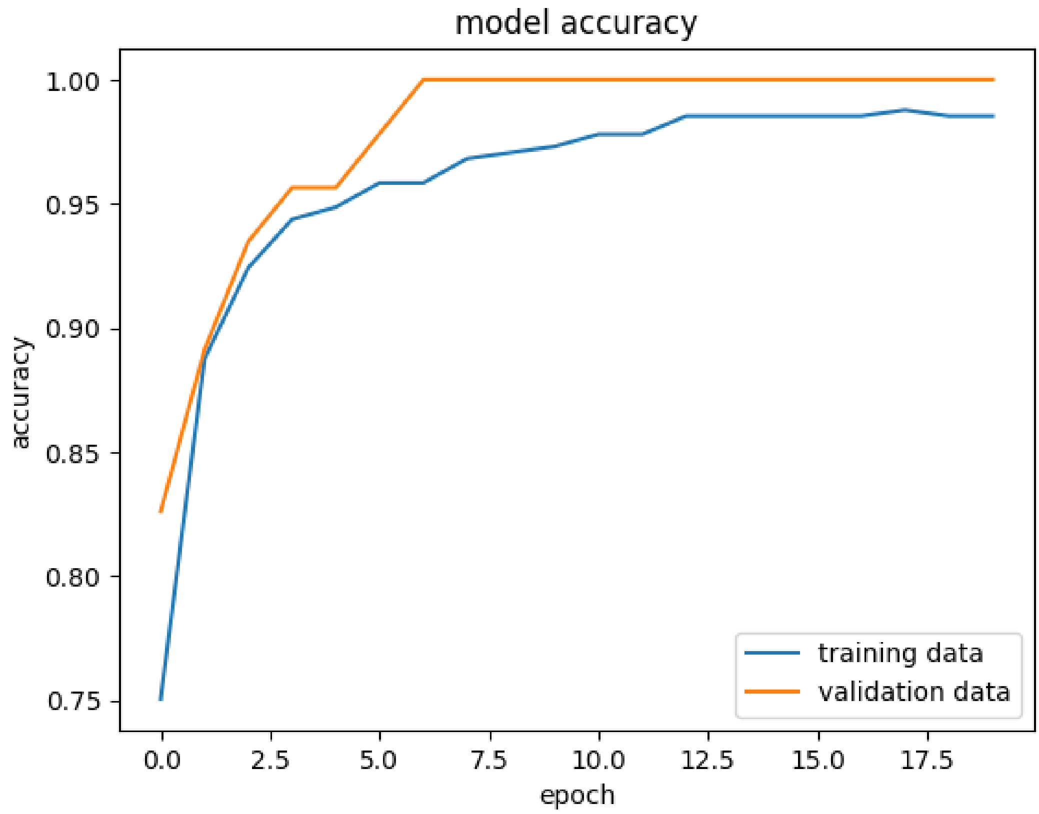
below are the plots for loss and accuracy vs epoch

In [68]:

```
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[68]:

```
<matplotlib.legend.Legend at 0x7c7d39fc24a0>
```



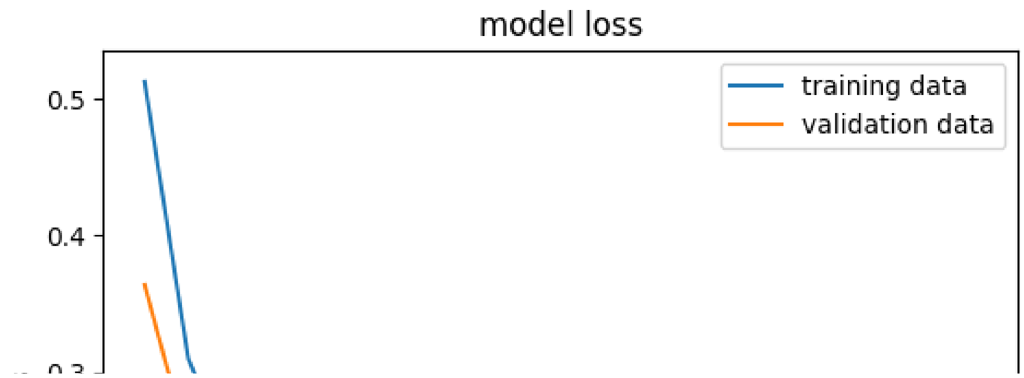
In [69]:

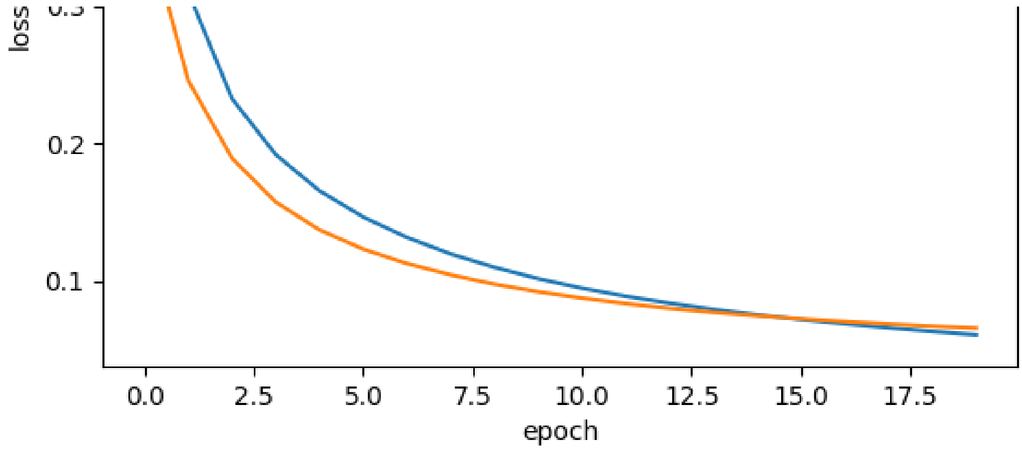
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training data', 'validation data'], loc = 'upper right')
```

Out[69]:

```
<matplotlib.legend.Legend at 0x7c7d49d41b70>
```





ACCURACY ON TEST DATA

In [70]:

```
loss, accuracy = model.evaluate(X_test_std, Y_test)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9794 - loss: 0.0762
```

In [71]:

```
print(accuracy)
```

```
0.9824561476707458
```

In [72]:

```
print(loss)
```

```
0.0646393746137619
```

In [73]:

```
print(X_test_std.shape)
```

```
(114, 30)
```

In [74]:

```
print(X_test_std[0])
```

```
[-0.4877952  -0.25088379 -0.46378664 -0.51543986  0.05784012 -0.0262922
 -0.10351188 -0.31929301  0.40877389  0.11968017 -0.0858903  -0.39144008
 -0.22090758 -0.2944085  -0.10670318 -0.38822597 -0.05805714 -0.18386347
 -0.30097441 -0.05389032 -0.29186744 -0.2492245  -0.34337233 -0.37693497
  0.32080655 -0.13570975  0.0219982  -0.21989091  0.23083947  0.17173277]
```

In [75]:

```
Y_pred = model.predict(X_test_std)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```

Here'model.predict' gives Prediction Probablity of each class for that data point

In [76]:

```
print(Y_pred.shape)
```

```
(114, 2)
```

In [77]:

```
print(Y_pred[0])
```

```
[0.36871758 0.90940666]
```

In [78]:

```
print(X_test_std)
```

```
[[-0.4877952  -0.25088379 -0.46378664 ... -0.21989091  0.23083947
   0.17173277]
 [ 1.37325734  0.36318719  1.29254295 ...  0.93019499 -0.58958803
  -0.972629  ]
 [ 0.37225845 -0.05148067  0.38772954 ...  0.52157274 -0.08280128
  -0.21536971]
 ...
 [-0.76105639 -1.08248546 -0.76208585 ... -0.30042664 -0.3893541
  -0.32657562]
 [ 0.01558068  1.84511495  0.00969106 ... -0.50681802 -1.76086788
  -0.33345979]
 [ 1.88813896  2.58607883  1.79246268 ...  0.73702811 -0.02786985
  -0.10998887]]
```

In [79]:

```
print(Y_pred)
```

```
[[3.68717581e-01 9.09406662e-01]
 [9.49734807e-01 7.82909710e-03]
 [7.29148269e-01 9.83393714e-02]
 [2.69956619e-01 9.90913510e-01]
 [1.73727691e-01 9.97892797e-01]
 [9.99541104e-01 5.48638927e-05]
 [9.93384242e-01 4.11466142e-04]
 [6.52125776e-01 2.39761755e-01]
 [6.35126650e-01 8.01352561e-01]
 [1.03908285e-01 9.93952036e-01]
 [3.97986621e-01 9.39810693e-01]
 [7.58947074e-01 1.29158661e-01]
 [2.69596845e-01 9.51986790e-01]
 [6.08443141e-01 2.40193516e-01]
 [2.14384705e-01 9.93139327e-01]
 [9.01194811e-01 2.61751581e-02]
 [1.35579452e-01 9.70830321e-01]
 [2.72782177e-01 9.99250591e-01]
 [3.12361091e-01 9.99953985e-01]
 [9.57143962e-01 4.34301095e-03]
 [3.58143598e-01 8.58646989e-01]
 [3.98802489e-01 9.81752574e-01]
 [9.96306062e-01 4.00131918e-04]
 [1.42152593e-01 9.99471426e-01]
 [2.04580054e-01 9.97151375e-01]
 [1.39170587e-01 9.84659612e-01]
 [1.44612283e-01 9.90306020e-01]
 [1.75287321e-01 9.94423091e-01]
 [2.51135588e-01 9.85127568e-01]
 [9.91486132e-01 4.28293226e-03]
 [2.44109750e-01 9.97621357e-01]
 [1.20472841e-01 9.97034967e-01]
 [2.45707691e-01 9.95579302e-01]
 [2.31070369e-01 9.63147998e-01]
 [2.97606260e-01 9.98362780e-01]
 [3.10764879e-01 9.87725317e-01]
 [6.89205289e-01 3.55033189e-01]
 [1.19845495e-01 9.84520197e-01]
 [9.30966496e-01 1.28668100e-02]
 [3.33247572e-01 8.59483004e-01]
 [2.32073113e-01 9.98748541e-01]
 [7.99586177e-01 6.81976974e-02]
 [2.84895897e-01 9.83313203e-01]
 [1.89191088e-01 9.94748175e-01]
 [2.87213981e-01 8.36745203e-01]
 [3.88636976e-01 9.70685661e-01]
 [4.43138897e-01 9.97547388e-01]
 [2.17191473e-01 9.98739541e-01]
 [5.35536528e-01 9.72234130e-01]
```

```
 [9.26552191e-02 9.91803646e-01]
 [9.15397942e-01 1.68135520e-02]
 [9.89276230e-01 1.70424394e-03]
 [7.37309039e-01 8.75177383e-01]
 [1.50190189e-01 8.90652359e-01]
 [1.58921897e-01 9.97007370e-01]
 [1.90719977e-01 9.76380706e-01]
 [2.18253106e-01 9.97604430e-01]
 [9.99399364e-01 2.21786922e-05]
 [5.73976874e-01 4.10342872e-01]
 [1.29326105e-01 9.97449756e-01]
 [3.00388932e-01 9.78869557e-01]
 [9.72675323e-01 4.45455639e-03]
 [9.77223098e-01 8.09118326e-04]
 [2.67274857e-01 9.39317882e-01]
 [1.95790946e-01 9.95521903e-01]
 [2.50207543e-01 9.56876695e-01]
 [9.48473930e-01 1.04553411e-02]
 [9.97850418e-01 8.08905519e-04]
 [3.81773025e-01 9.95706320e-01]
 [3.78076702e-01 9.55845773e-01]
 [8.34117353e-01 1.15504995e-01]
 [9.32898521e-01 7.01000020e-02]
 [1.97022721e-01 9.81200039e-01]
 [8.39150906e-01 6.50138780e-02]
 [1.82290182e-01 9.99854863e-01]
 [3.76961142e-01 9.57012296e-01]
 [2.73487031e-01 9.37870979e-01]
 [5.60811222e-01 6.47986591e-01]
 [1.01737000e-01 9.99294698e-01]
 [4.64193881e-01 9.74773765e-01]
 [8.67051184e-01 9.24002752e-02]
 [1.46397665e-01 9.99150157e-01]
 [5.58540404e-01 3.93063664e-01]
 [9.76572633e-01 1.33499876e-03]
 [8.52360785e-01 6.32489771e-02]
 [7.72722006e-01 1.06965967e-01]
 [9.91553724e-01 1.03936143e-01]
 [8.43380690e-01 4.01753820e-02]
 [4.21411358e-02 9.98379052e-01]
 [2.66970366e-01 9.64466214e-01]
 [3.62523645e-01 9.90390599e-01]
 [6.35508895e-01 7.32306659e-01]
 [2.54293740e-01 7.98824787e-01]
 [1.93164989e-01 9.98160183e-01]
 [5.32054484e-01 9.95406032e-01]
 [2.59332120e-01 9.97613966e-01]
 [9.69400644e-01 2.31236150e-03]
 [9.85464156e-01 9.27838776e-03]
 [1.61653757e-01 9.97992814e-01]
 [9.35303867e-01 1.99486967e-02]
 [9.08898532e-01 6.85537681e-02]
 [1.22395575e-01 9.99907076e-01]
 [9.53924596e-01 3.81063446e-02]
 [8.63132238e-01 3.60085070e-02]
 [2.75752455e-01 9.90567207e-01]
 [4.25820380e-01 9.32571173e-01]
 [2.46723190e-01 9.80000079e-01]
 [9.97015119e-01 2.53661594e-04]
 [3.17434072e-01 8.59048128e-01]
 [3.43554467e-01 9.16883230e-01]
 [7.76087940e-01 8.72754976e-02]
 [1.99169278e-01 9.91953909e-01]
 [5.47413409e-01 7.88669765e-01]
 [9.99077260e-01 2.87298190e-05]]
```

NOW USING 'argmax' FUNCTION

In [80]:

```
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
```

NOW CONVERT PREDICTION PROBABLITY TO CLASS LABELS

In [81]:

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

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

BUILDING THE PREDICTIVE SYSTEM

In [82]:

```
input_data = (11.76,21.6,74.72,427.9,0.08637,0.04966,0.01657,0.01115,0.1495,0.05888,0.40
62,1.21,2.635,28.47,0.005857,0.009758,0.01168,0.007445,0.02406,0.001769,12.98,25.72,82.98
,516.5,0.1085,0.08615,0.05523,0.03715,0.2433,0.06563)

#CONVERT INPUT DATA INTO NUMPY ARRAY
input_data_as_numpy_array = np.asarray(input_data)

#RESHAPE THE NUMPY ARRAY AS WE ARE PREDICTING FOR ONE DATA POINT
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

#STANDARDIZING THE INPUT DATA
input_data_std = scaler.transform(input_data_reshaped)

prediction = model.predict(input_data_std)
print(prediction)

prediction_label = [np.argmax(prediction)]
print(prediction_label)

if(prediction_label[0] == 0):
  print('The tumor is Malignant')

else:
  print('The tumor is Benign')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
[[0.19964646 0.98835504]]
[1]
The tumor is Benign
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X
does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

Conclusion

The project successfully built a neural network to classify breast tumors as malignant or benign using the Breast Cancer Wisconsin Dataset. The preprocessing of data, standardizing, and using a neural network model helped us achieve a high accuracy in classification. The performance of the model clearly showed the inverse relationship between loss and accuracy, as expected, where the loss decreased and accuracy increased in each epoch.

The use of StandardScaler ensured the data was standardized, which helped improve the model's convergence and overall performance. The project shows the effectiveness of machine learning, especially neural networks, in assisting medical diagnosis, particularly for early detection of cancerous cases of the breast, and also points

out the importance of preprocessing the data to get reliable and precise results.