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

```
# loading the data from sklearn
breast_cancer_dataset = sklearn.datasets.load_breast_cancer()
```

```
print(breast_cancer_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, 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_names': array(['malignant', 'benign'], dtyp
        '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_cancer.csv', 'data_module': 'sklearn.dataset
```

```
# loading the data to a data frame
data_frame = pd.DataFrame(breast_cancer_dataset.data, columns = breast_cancer_dataset.feature_names)
```

```
# print the first 5 rows of the dataframe
data_frame.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	wo perime
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152

5 rows × 30 columns

```
# adding the 'target' column to the data frame
data_frame['label'] = breast_cancer_dataset.target
```

```
# print last 5 rows of the dataframe
data_frame.tail()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	w
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2(
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	:

5 rows × 31 columns

```
# number of rows and columns in the dataset
data_frame.shape
```

```
(569, 31)
```

```
# getting some information about the data
data_frame.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
```

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

	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
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

```
# statistical measures about the data
data_frame.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	2
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	1
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	2
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	2
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	2
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	4

8 rows × 31 columns

```
# checking the distribution of Target Varibale
data_frame['label'].value_counts()
```

label	count
1	357
0	212

dtype: int64

1 --> Benign

0 --> Malignant

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

label	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius
0	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775	0.087990	0.192909	0.062680	...	21.134811
1	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058	0.025717	0.174186	0.062867	...	13.379801

2 rows × 30 columns

Separating the features and target

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

```
print(X)
```

```
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
```

```
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]
```

```
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
```

Splitting the data into training data & Testing data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

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

Standardize the data

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

```
scaler = StandardScaler()

X_train_std = scaler.fit_transform(X_train)

X_test_std = scaler.transform(X_test)
```

```
# importing tensorflow and Keras
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
```

```
# setting up the layers of Neural Network

model = keras.Sequential([
                          keras.layers.Flatten(input_shape=(30,)),
                          keras.layers.Dense(20, activation='relu'),
                          keras.layers.Dense(2, activation='sigmoid')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(**kwargs)
```

```
# compiling the Neural Network

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

```
# training the Meural Network

history = model.fit(X_train_std, Y_train, validation_split=0.1, epochs=10)
```

```
Epoch 1/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 3s 92ms/step - accuracy: 0.6854 - loss: 0.5907 - val_accuracy: 0.8913 - val_loss: 0.4292
Epoch 2/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8555 - loss: 0.4023 - val_accuracy: 0.9348 - val_loss: 0.3120
Epoch 3/10
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9162 - loss: 0.3016 - val_accuracy: 0.9348 - val_loss: 0.2466
Epoch 4/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9410 - loss: 0.2435 - val_accuracy: 0.9348 - val_loss: 0.2047
Epoch 5/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9494 - loss: 0.2053 - val_accuracy: 0.9565 - val_loss: 0.1760
Epoch 6/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9509 - loss: 0.1777 - val_accuracy: 0.9565 - val_loss: 0.1555
Epoch 7/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9598 - loss: 0.1567 - val_accuracy: 0.9565 - val_loss: 0.1397
Epoch 8/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9664 - loss: 0.1402 - val_accuracy: 0.9565 - val_loss: 0.1275
Epoch 9/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9787 - loss: 0.1269 - val_accuracy: 0.9565 - val_loss: 0.1179
Epoch 10/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9799 - loss: 0.1160 - val_accuracy: 0.9565 - val_loss: 0.1104
```

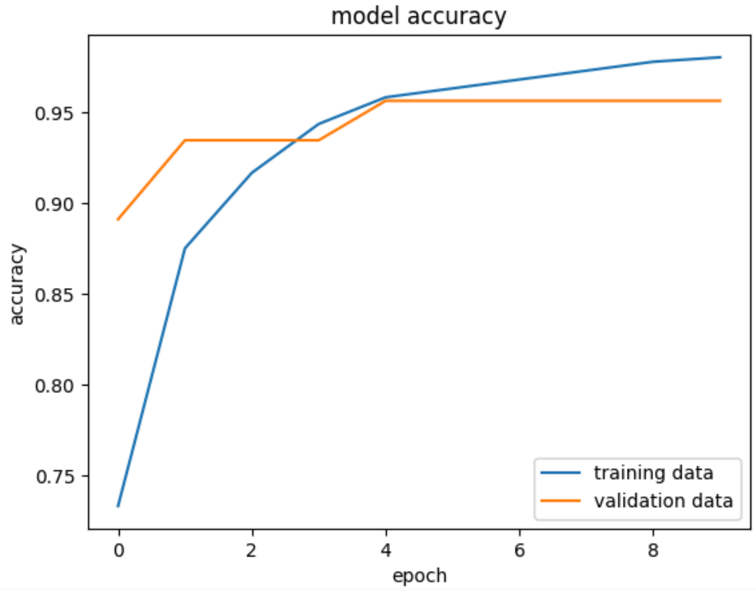
Visualizing accuracy and loss

```
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')
```

```
<matplotlib.legend.Legend at 0x79c57bdaee10>
```

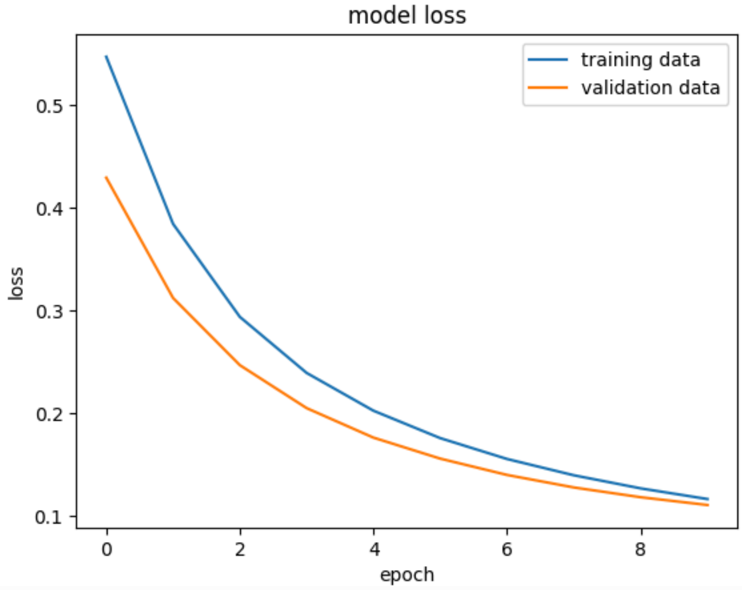



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

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

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

```
<matplotlib.legend.Legend at 0x79c57bd16710>
```



Accuracy of the model on test data

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

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.9651 - loss: 0.1069
0.9649122953414917
```

```
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 ]
```

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

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```

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

```
(114, 2)
[0.4896841 0.9292942]
```

```
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]]
```

```
print(Y_pred)
```

```
 [9.58952308e-01 7.13354200e-02]
 [8.43964756e-01 9.54346120e-01]
 [9.77803469e-01 4.40334156e-03]
 [9.10227776e-01 1.21170156e-01]
 [1.74297720e-01 5.49629927e-01]
 [8.37107122e-01 8.78640532e-01]
 [7.68183649e-01 1.95523232e-01]
 [9.84420776e-01 1.00305500e-02]
 [1.67858273e-01 8.17001462e-01]
 [6.49039745e-01 1.63872615e-01]
 [1.36969797e-02 8.28537226e-01]
 [5.98057985e-01 1.51040152e-01]
 [7.00484589e-02 8.05805683e-01]
 [3.99227440e-02 6.73314631e-01]
 [4.47756648e-01 6.85958683e-01]
 [4.93220061e-01 3.00570101e-01]
 [9.32083488e-01 2.00960711e-02]
 [8.17510307e-01 6.88179731e-02]
 [6.50133431e-01 2.82511003e-02]
 [3.53340119e-01 8.81022215e-01]
 [1.16706394e-01 7.64771700e-01]
 [6.73648298e-01 5.15590549e-01]
 [5.92761576e-01 9.93173778e-01]
 [1.07719868e-01 8.56152952e-01]
 [2.85010040e-01 6.86984181e-01]
 [9.97849107e-01 2.18055421e-03]
 [1.14024587e-01 7.98489571e-01]
 [1.26894057e-01 4.78090793e-01]
 [2.67476737e-01 9.80593204e-01]
 [9.90631044e-01 5.36604449e-02]
 [5.90171814e-01 1.86554804e-01]
 [1.06074184e-01 7.13545382e-01]
 [9.04580951e-01 2.11445596e-02]
 [8.32978666e-01 3.29208411e-02]
 [1.52823925e-01 4.92380500e-01]
 [2.48545054e-02 9.09400821e-01]
 [1.19159985e-02 8.81372392e-01]
 [6.93197668e-01 1.65580690e-01]
 [9.98948276e-01 7.55111570e-04]
 [9.69142318e-01 2.45781196e-03]
 [1.08432710e-01 5.93069017e-01]
 [5.02252802e-02 9.03824151e-01]
 [8.86870176e-03 9.85297859e-01]
 [8.86030272e-02 9.05318499e-01]
 [5.41608632e-02 9.95704234e-01]
 [3.22860926e-01 7.14575410e-01]
 [9.60577726e-01 1.57558974e-02]
 [9.36341941e-01 6.77333912e-03]
 [6.56910717e-01 4.83646363e-01]
 [6.92940235e-01 7.82594830e-02]]
```

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

```
#  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
```

```
# converting the prediction probability to class labels

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

```
[np.int64(1), np.int64(1), np.int64(1), np.int64(0), np.int64(1), np.int64(0), np.int64(1), np.int64(1), np.int64(1), np.int64(1), n
```

Building the predictive system

```
input_data = (11.76,21.6,74.72,427.9,0.08637,0.04966,0.01657,0.01115,0.1495,0.05888,0.4062,1.21,2.635,28.47,0.005857,0.009758,0.01168,0

# change the input_data to a 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 129ms/step
[[0.02820272 0.8278283 ]]
```