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

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

print(breast_cancer_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()
```

```
{'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, 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, 'tar
        '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': 'brea
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	sy
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	

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()

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

# getting some information about the data
data_frame.info()

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

# statistical measures about the data
data_frame.describe()

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

```
<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
1    357
0    212
Name: label, dtype: int64
```

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

label	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity
0	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775
1	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058

2 rows × 30 columns

```
# Separating the features and target
X = data_frame.drop(columns='label', axis=1)
Y = data_frame['label']

print(X)

print(Y)
```

```
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]
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)
```



```
# 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 necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

# set random seed for reproducibility
tf.random.set_seed(3)

# setting up the layers of the neural network
model = Sequential([
    # Input layer: Flatten layer to convert input shape (30,) to a 1D array
    Dense(128, input_shape=(30,), activation='relu'),

    # Hidden layers: Adding more dense layers with relu activation
    Dense(256, activation='relu'),
    Dropout(0.5),  # Dropout layer to prevent overfitting by randomly dropping 50% of neurons

    Dense(128, activation='relu'),
    BatchNormalization(),  # Batch normalization to stabilize and accelerate the training process

    Dense(64, activation='relu'),
    Dropout(0.4),  # Dropout layer to prevent overfitting by randomly dropping 40% of neurons

    Dense(32, activation='relu'),
    BatchNormalization(),

    Dense(16, activation='relu'),
    Dropout(0.3),  # Dropout layer to prevent overfitting by randomly dropping 30% of neurons

    # Output layer: Dense layer with softmax activation for multi-class classification
    Dense(2, activation='softmax')
])



# Early stopping to prevent overfitting and save training time
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Print model summary
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 128)               3968

 dense_19 (Dense)            (None, 256)               33024

 dropout_6 (Dropout)         (None, 256)               0

 dense_20 (Dense)            (None, 128)               32896

 batch_normalization_4 (Bat  (None, 128)               512
 chNormalization)

 dense_21 (Dense)            (None, 64)                8256

 dropout_7 (Dropout)         (None, 64)                0

 dense_22 (Dense)            (None, 32)                2080

 batch_normalization_5 (Bat  (None, 32)                128
 chNormalization)

 dense_23 (Dense)            (None, 16)                528

 dropout_8 (Dropout)         (None, 16)                0

 dense_24 (Dense)            (None, 2)                 34

=================================================================
Total params: 81426 (318.07 KB)
Trainable params: 81106 (316.82 KB)
Non-trainable params: 320 (1.25 KB)
_________________________________________________________________
```

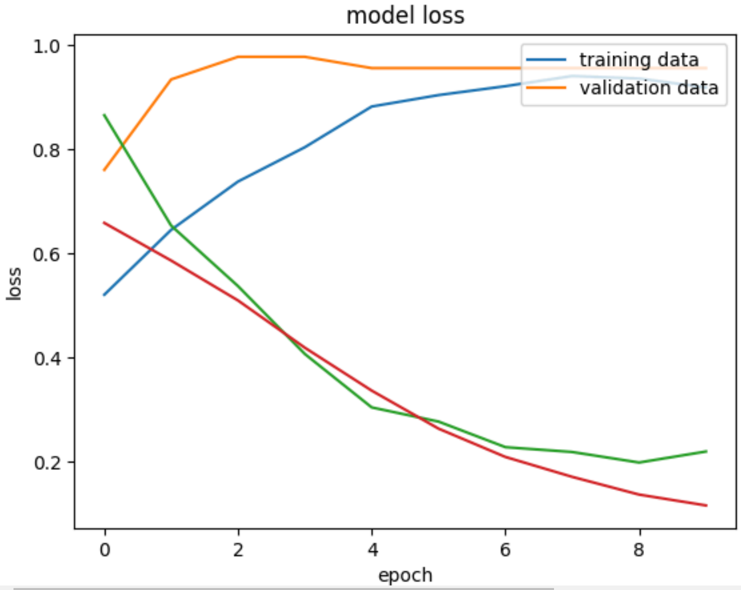
```
# Compile the model
model.compile(optimizer='adam',  # Using Adam optimizer for optimization
              loss='sparse_categorical_crossentropy',  # Sparse categorical crossentropy for multi-class classification
              metrics=['accuracy'])  # Monitoring accuracy during training


# Early stopping to prevent overfitting and save training time
```

```
# Early stopping to prevent overfitting and save training time
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Print model summary
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 128)               3968

 dense_19 (Dense)            (None, 256)               33024

 dropout_6 (Dropout)         (None, 256)               0

 dense_20 (Dense)            (None, 128)               32896

 batch_normalization_4 (Bat  (None, 128)               512
 chNormalization)

 dense_21 (Dense)            (None, 64)                8256

 dropout_7 (Dropout)         (None, 64)                0

 dense_22 (Dense)            (None, 32)                2080

 batch_normalization_5 (Bat  (None, 32)                128
 chNormalization)

 dense_23 (Dense)            (None, 16)                528

 dropout_8 (Dropout)         (None, 16)                0

 dense_24 (Dense)            (None, 2)                 34

=================================================================
Total params: 81426 (318.07 KB)
Trainable params: 81106 (316.82 KB)
Non-trainable params: 320 (1.25 KB)
_________________________________________________________________
```

```
# training the Meural Network

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

"""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')

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')
```

```
Epoch 1/10
13/13 [==============================] - 6s 74ms/step - loss: 0.8659 - accuracy: 0.52
Epoch 2/10
13/13 [==============================] - 0s 16ms/step - loss: 0.6536 - accuracy: 0.64
Epoch 3/10
13/13 [==============================] - 0s 14ms/step - loss: 0.5373 - accuracy: 0.73
Epoch 4/10
13/13 [==============================] - 0s 12ms/step - loss: 0.4067 - accuracy: 0.80
Epoch 5/10
13/13 [==============================] - 0s 16ms/step - loss: 0.3040 - accuracy: 0.88
Epoch 6/10
13/13 [==============================] - 0s 12ms/step - loss: 0.2768 - accuracy: 0.90
Epoch 7/10
13/13 [==============================] - 0s 13ms/step - loss: 0.2276 - accuracy: 0.92
Epoch 8/10
13/13 [==============================] - 0s 10ms/step - loss: 0.2183 - accuracy: 0.94
Epoch 9/10
13/13 [==============================] - 0s 17ms/step - loss: 0.1980 - accuracy: 0.93
Epoch 10/10
13/13 [==============================] - 0s 16ms/step - loss: 0.2192 - accuracy: 0.91
<matplotlib.legend.Legend at 0x7e953f3435e0>
```




```
"""Accuracy of the model on test data"""

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

print(X_test_std.shape)
print(X_test_std[0])

Y_pred = model.predict(X_test_std)

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

print(X_test_std)

print(Y_pred)
```

```
 [1.25814572e-01 8.74185443e-01]
 [8.49285483e-01 1.50714591e-01]
 [9.84075069e-01 1.59248207e-02]
 [9.07692552e-01 9.23074260e-02]
 [9.83973026e-01 1.60268676e-02]
 [1.45389453e-01 8.54610562e-01]
 [8.08890164e-02 9.19110954e-01]
 [6.11699462e-01 3.88300449e-01]
 [3.56241278e-02 9.64375913e-01]
 [4.90115248e-02 9.50988472e-01]
 [1.09685376e-01 8.90314639e-01]
 [9.98139143e-01 1.86083699e-03]
 [6.37413710e-02 9.36258674e-01]
 [1.07791111e-01 8.92208934e-01]
 [3.01897135e-02 9.69810307e-01]
 [9.75657582e-01 2.43424233e-02]
 [8.72165024e-01 1.27834946e-01]
 [7.38479868e-02 9.26151991e-01]
 [9.90186572e-01 9.81338881e-03]
 [9.84424412e-01 1.55756120e-02]
 [9.40662175e-02 9.05933797e-01]
 [3.01950313e-02 9.69804883e-01]
 [2.73890477e-02 9.72610891e-01]
 [8.59931409e-01 1.40068516e-01]
 [9.99123454e-01 8.76450446e-04]
 [9.98843610e-01 1.15647586e-03]
 [8.35522860e-02 9.16447759e-01]
 [5.57413511e-02 9.44258630e-01]
 [4.07444686e-02 9.59255457e-01]
 [8.63488615e-02 9.13651109e-01]
 [3.05610374e-02 9.69438970e-01]
 [9.69271585e-02 9.03072834e-01]
 [9.91290748e-01 8.70919414e-03]
 [9.91175115e-01 8.82485323e-03]
 [3.04033577e-01 6.95966423e-01]
 [9.63445723e-01 3.65542807e-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)
```

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

```
"""**Building the predictive system**"""

input_data = (11.76,21.6,74.72,427.9,0.08637,0.04966,0.01657,0.01115,0.1495,0.05888,0.4062,1.21,2.635,28.47,0.005857,0.009758,0.01168,0.0


# 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)
```

```
1/1 [==============================] - 0s 20ms/step
[[0.04395929 0.9560407 ]]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler wa
  warnings.warn(
```

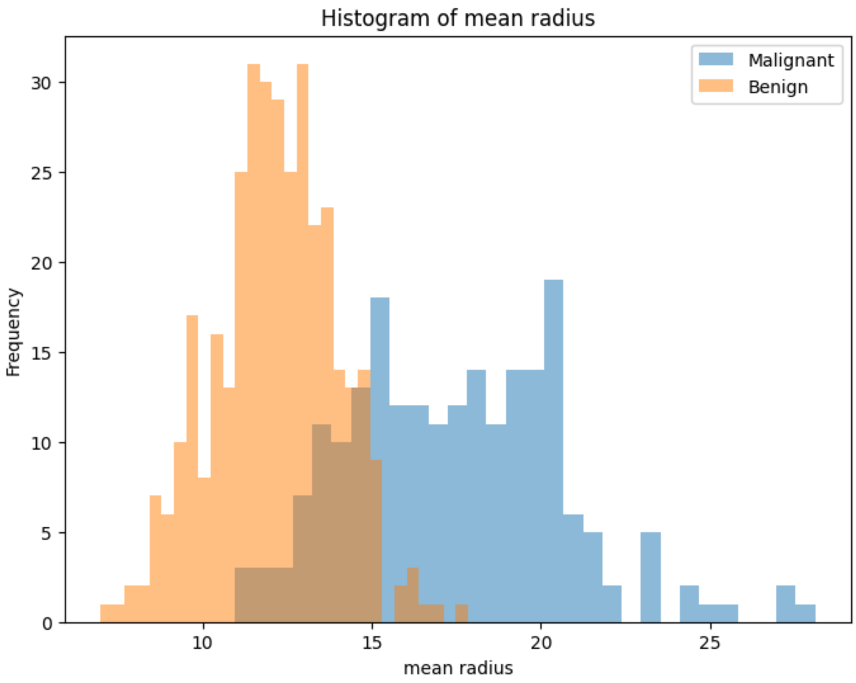
```
prediction_label = [np.argmax(prediction)]
print(prediction_label)
```

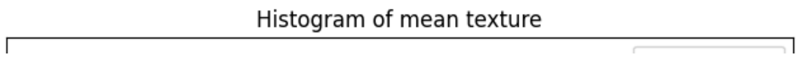
```
[1]
```

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

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

```
The tumor is Benign
```

```
# Plot histograms for a few features
features_to_plot = ['mean radius', 'mean texture', 'mean perimeter']

for feature in features_to_plot:
    plt.figure(figsize=(8, 6))
    plt.hist(data_frame[data_frame['label'] == 0][feature], bins=30, alpha=0.5, label='Malignant')
    plt.hist(data_frame[data_frame['label'] == 1][feature], bins=30, alpha=0.5, label='Benign')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.title(f'Histogram of {feature}')
    plt.legend()
    plt.show()
```


Histogram of mean radius
Malignant
Benign
Frequency
0
5
10
15
20
25
30
10
15
20
25
mean radius


Histogram of mean texture