**IMPORTING THE DEPENDECIES**

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

Double-click (or enter) to edit

**Data Collection & Processing**

```
# 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(['malignan
        '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
```

```
# loding 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 data frame
data_frame.head()
```

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | co |
|---|---|---|---|---|---|---|---|
| **0** | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | |
| **1** | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | |
| **2** | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | |
| **3** | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | |
| **4** | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | |

```
# 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 |
|---|---|---|---|---|---|---|
| **564** | 21.56 | 22.39 | 142.00 | 1479.0 | 0.11100 | 0.11590 |
| **565** | 20.13 | 28.25 | 131.20 | 1261.0 | 0.09780 | 0.10340 |
| **566** | 16.60 | 28.08 | 108.30 | 858.1 | 0.08455 | 0.10230 |
| **567** | 20.60 | 29.33 | 140.10 | 1265.0 | 0.11780 | 0.27700 |
| **568** | 7.76 | 24.54 | 47.92 | 181.0 | 0.05263 | 0.04362 |

```
# 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()
```

```
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 |
|---|---|---|---|---|---|
| **count** | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 |
| **mean** | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096360 |
| **std** | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014064 |
| **min** | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052630 |
| **25%** | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086370 |
| **50%** | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095870 |
| **75%** | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105300 |

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

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

1 --> Benign

0 --> Malignant

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

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | co |
|---|---|---|---|---|---|---|
| **label** | | | | | | |
| **0** | 17.462830 | 21.604906 | 115.365377 | 978.376415 | 0.102898 | |
| **1** | 12.146524 | 17.914762 | 78.075406 | 462.790196 | 0.092478 | |

Separating the features and target

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

```
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
```

```
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**

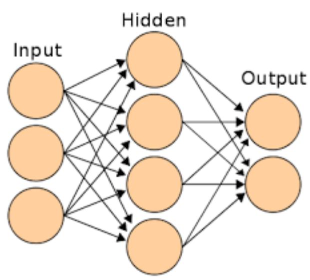
```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()

X_train_std = scaler.fit_transform(X_train)

X_test_std = scaler.transform(X_test)
```

**Building the Neural Network**




```
# 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')
])
```

```
# 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 [==============================] - 1s 20ms/step - loss: 0.5694 - accuracy: 0.7139 - val_loss: 0.3319 - val_
Epoch 2/10
13/13 [==============================] - 0s 4ms/step - loss: 0.3568 - accuracy: 0.8729 - val_loss: 0.2213 - val_a
Epoch 3/10
13/13 [==============================] - 0s 6ms/step - loss: 0.2602 - accuracy: 0.9218 - val_loss: 0.1736 - val_a
Epoch 4/10
13/13 [==============================] - 0s 5ms/step - loss: 0.2154 - accuracy: 0.9291 - val_loss: 0.1468 - val_a
Epoch 5/10
13/13 [==============================] - 0s 6ms/step - loss: 0.1844 - accuracy: 0.9438 - val_loss: 0.1301 - val_a
Epoch 6/10
13/13 [==============================] - 0s 4ms/step - loss: 0.1638 - accuracy: 0.9535 - val_loss: 0.1182 - val_a
Epoch 7/10
13/13 [==============================] - 0s 4ms/step - loss: 0.1477 - accuracy: 0.9609 - val_loss: 0.1091 - val_a
Epoch 8/10
```

```
13/13 [==============================] - 0s 4ms/step - loss: 0.1348 - accuracy: 0.9707 - val_loss: 0.1016 - val_a
Epoch 9/10
13/13 [==============================] - 0s 5ms/step - loss: 0.1242 - accuracy: 0.9731 - val_loss: 0.0950 - val_a
Epoch 10/10
13/13 [==============================] - 0s 5ms/step - loss: 0.1147 - accuracy: 0.9780 - val_loss: 0.0904 - val_a
```

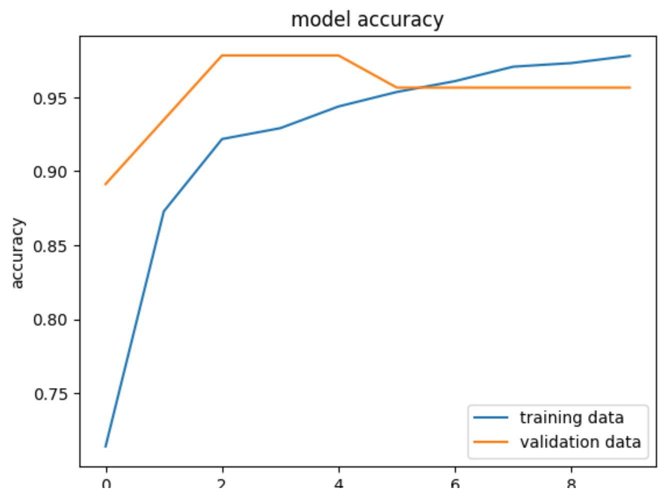
Visualizing accuracy and loss

```
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 0x786b214ebd90>
```

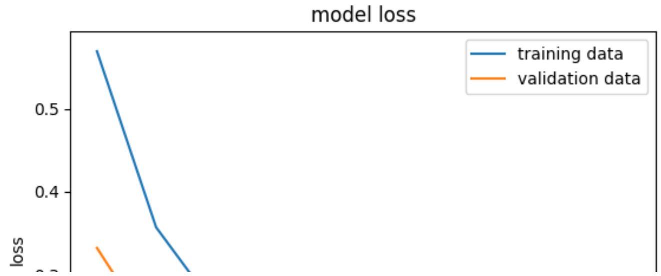



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

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

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

```
<matplotlib.legend.Legend at 0x786b214e85b0>
```




Accuracy of the model on test data

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

```
4/4 [==============================] - 0s 3ms/step - loss: 0.1206 - accuracy: 0.9649
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 2ms/step
```

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

```
(114, 2)
[0.5910623  0.64777416]
```

```
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)
```

```
[[0.5910623  0.64777416]
 [0.57262623 0.53137416]
 [0.08178474 0.92251354]
 [0.99999636 0.8754512 ]
 [0.6494268  0.6795414 ]
 [0.9978378  0.73924863]
 [0.35568988 0.65768343]
 [0.06551942 0.89959633]
```

```
[0.1704827  0.8185148 ]
[0.11982249 0.89849025]
[0.5166799  0.47550526]
[0.2432145  0.73547703]
[0.3054033  0.73301727]
[0.32677945 0.6794893 ]
[0.11260096 0.8842174 ]
[0.9769998  0.47413704]
[0.08762719 0.8973993 ]
[0.16068268 0.94328135]
[0.14915735 0.7968978 ]
[0.99748766 0.70721126]
[0.94598246 0.8555197 ]
[0.066418   0.9159178 ]
[0.11691153 0.8489473 ]
[0.07170216 0.93611085]
[0.19703601 0.7964921 ]
[0.9919995  0.5575889 ]
[0.15023275 0.7869088 ]
[0.26323164 0.7787339 ]
[0.9899322  0.44185722]
[0.9872928  0.408697  ]
[0.20459795 0.849198  ]
[0.19734438 0.8272765 ]
[0.14040704 0.87954384]
[0.99972373 0.14902125]
[0.9964441  0.6718576 ]
[0.18507044 0.8106896 ]
[0.02255836 0.8897489 ]
[0.23570876 0.5984493 ]
[0.05192166 0.92064077]
[0.22142278 0.8514634 ]
[0.9999666  0.88097465]
[0.87940365 0.48971888]
[0.00588301 0.69957805]
[0.21378544 0.89768004]
[0.94594187 0.6020921 ]
[0.11570267 0.8739163 ]
[0.09937078 0.9309568 ]
[0.08318587 0.90201384]
[0.9996934  0.2658929 ]
[0.97562253 0.5196244 ]
[0.13347326 0.90047187]
[0.9149535  0.7256508 ]
[0.36942926 0.5317479 ]
[0.08153431 0.87005514]
[0.10065925 0.9040531 ]
[0.7661681  0.70631254]
[0.07776804 0.7460908 ]
[0.14298692 0.8882097 ]
```

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, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 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.

# 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 54ms/step
[[0.03737957 0.8517287 ]]
[1]
The tumor is Benign
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, bu
  warnings.warn(
```

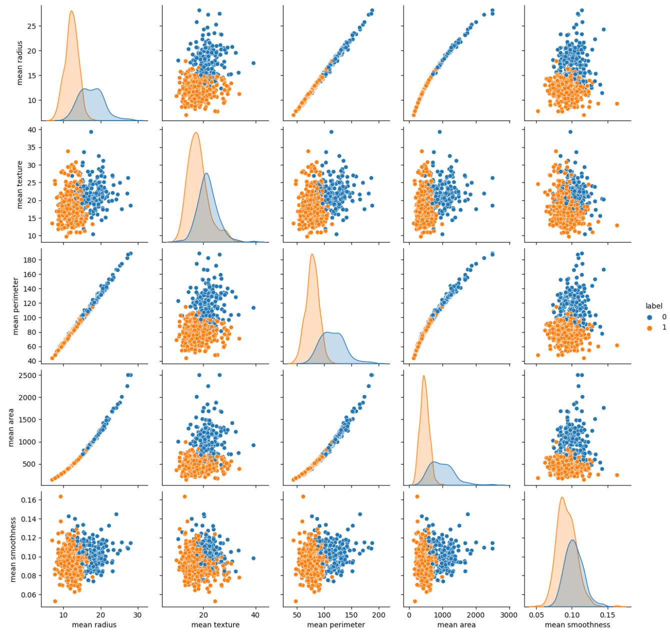
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.datasets
from sklearn.model_selection import train_test_split
```

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

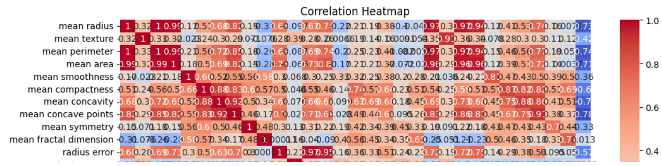
```
# loading the data to a data frame
data_frame = pd.DataFrame(breast_cancer_dataset.data, columns=breast_cancer_dataset.feature_names)
```

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

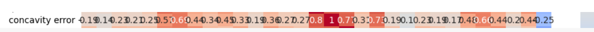
```
# Visualization: Pairplot
sns.pairplot(data_frame, hue='label', vars=breast_cancer_dataset.feature_names[:5])
plt.show()
```



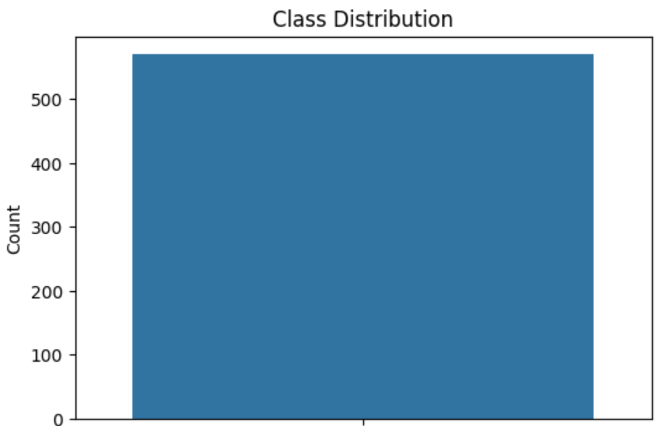

```
plt.figure(figsize=(12, 8))
sns.heatmap(data_frame.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```




**Data analysis - Class distribution**



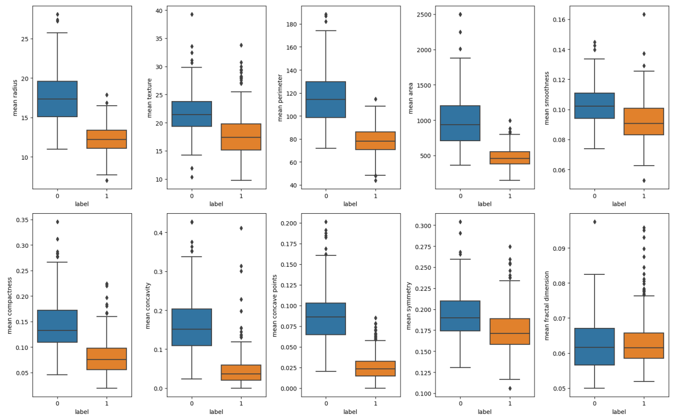

```
plt.figure(figsize=(6, 4))
sns.countplot(data_frame['label'])
plt.title('Class Distribution')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```




**Data analysis - Mean feature comparison between classes**

Double-click (or enter) to edit

```
mean_features = breast_cancer_dataset.feature_names[:10]
plt.figure(figsize=(16, 10))
for i, feature in enumerate(mean_features):
    plt.subplot(2, 5, i + 1)
    sns.boxplot(x='label', y=feature, data=data_frame)
plt.tight_layout()
plt.show()
```




Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit