```
import keras
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('/content/wdbc.data')
df
```

	842302	M	17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	...	2
0	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	24
1	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	23
2	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	14
3	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	22
4	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	...	15
...	...	...	...	...	...	...	...	...	...	...	...	
563	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	25
564	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	23
565	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	18
566	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	25
567	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	9

568 rows × 32 columns

```
def dataSetAnalysis(df):
    print("Dataset Head")
    print(df.head(3))

    print("Dataset Features")
    print(df.columns.values)

    print("Dataset Features Details")
    print(df.info())

    print("Dataset Numerical Features")
    print(df.describe())

    print("Dataset Categorical Features")
    print(df.describe(include=['O']))

dataSetAnalysis(df)
```

```
Dataset Head
     842302  M  17.99  10.38   122.8    1001   0.1184   0.2776  0.3001  \
0    842517  M  20.57  17.77  132.90  1326.0  0.08474  0.07864  0.0869
1  84300903  M  19.69  21.25  130.00  1203.0  0.10960  0.15990  0.1974
2  84348301  M  11.42  20.38   77.58   386.1  0.14250  0.28390  0.2414

    0.1471  ...  25.38  17.33   184.6    2019  0.1622  0.6656  0.7119  0.2654  \
0  0.07017  ...  24.99  23.41  158.80  1956.0  0.1238  0.1866  0.2416  0.1860
1  0.12790  ...  23.57  25.53  152.50  1709.0  0.1444  0.4245  0.4504  0.2430
2  0.10520  ...  14.91  26.50   98.87   567.7  0.2098  0.8663  0.6869  0.2575

   0.4601   0.1189
0  0.2750  0.08902
1  0.3613  0.08758
2  0.6638  0.17300

[3 rows x 32 columns]
Dataset Features
['842302' 'M' '17.99' '10.38' '122.8' '1001' '0.1184' '0.2776' '0.3001'
 '0.1471' '0.2419' '0.07871' '1.095' '0.9053' '8.589' '153.4' '0.006399'
 '0.04904' '0.05373' '0.01587' '0.03003' '0.006193' '25.38' '17.33'
 '184.6' '2019' '0.1622' '0.6656' '0.7119' '0.2654' '0.4601' '0.1189']
Dataset Features Details
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568 entries, 0 to 567
Data columns (total 32 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
```

```
 0   842302    568 non-null    int64  
 1   M         568 non-null    object 
 2   17.99     568 non-null    float64
 3   10.38     568 non-null    float64
 4   122.8     568 non-null    float64
 5   1001      568 non-null    float64
 6   0.1184    568 non-null    float64
 7   0.2776    568 non-null    float64
 8   0.3001    568 non-null    float64
 9   0.1471    568 non-null    float64
 10  0.2419    568 non-null    float64
 11  0.07871   568 non-null    float64
 12  1.095     568 non-null    float64
 13  0.9053    568 non-null    float64
 14  8.589     568 non-null    float64
 15  153.4     568 non-null    float64
 16  0.006399  568 non-null    float64
 17  0.04904   568 non-null    float64
 18  0.05373   568 non-null    float64
 19  0.01587   568 non-null    float64
 20  0.03003   568 non-null    float64
 21  0.006193  568 non-null    float64
 22  25.38     568 non-null    float64
 23  17.33     568 non-null    float64
 24  184.6     568 non-null    float64
 25  2019      568 non-null    float64
 26  0.1622    568 non-null    float64
 27  0.6656    568 non-null    float64
 28  0.7119    568 non-null    float64
 29  0.2654    568 non-null    float64
```

```
X = df.iloc[:,2:32]
y = df.iloc[:,1]
print(X)
print(y)
```

```
     17.99  10.38   122.8    1001   0.1184   0.2776   0.3001   0.1471  0.2419  \
0    20.57  17.77  132.90  1326.0  0.08474  0.07864  0.08690  0.07017  0.1812   
1    19.69  21.25  130.00  1203.0  0.10960  0.15990  0.19740  0.12790  0.2069   
2    11.42  20.38   77.58   386.1  0.14250  0.28390  0.24140  0.10520  0.2597   
3    20.29  14.34  135.10  1297.0  0.10030  0.13280  0.19800  0.10430  0.1809   
4    12.45  15.70   82.57   477.1  0.12780  0.17000  0.15780  0.08089  0.2087   
..     ...    ...     ...     ...      ...      ...      ...      ...     ...   
563  21.56  22.39  142.00  1479.0  0.11100  0.11590  0.24390  0.13890  0.1726   
564  20.13  28.25  131.20  1261.0  0.09780  0.10340  0.14400  0.09791  0.1752   
565  16.60  28.08  108.30   858.1  0.08455  0.10230  0.09251  0.05302  0.1590   
566  20.60  29.33  140.10  1265.0  0.11780  0.27700  0.35140  0.15200  0.2397   
567   7.76  24.54   47.92   181.0  0.05263  0.04362  0.00000  0.00000  0.1587   

     0.07871  ...   25.38  17.33   184.6    2019   0.1622   0.6656  0.7119  \
0    0.05667  ...  24.990  23.41  158.80  1956.0  0.12380  0.18660  0.2416   
1    0.05999  ...  23.570  25.53  152.50  1709.0  0.14440  0.42450  0.4504   
2    0.09744  ...  14.910  26.50   98.87   567.7  0.20980  0.86630  0.6869   
3    0.05883  ...  22.540  16.67  152.20  1575.0  0.13740  0.20500  0.4000   
4    0.07613  ...  15.470  23.75  103.40   741.6  0.17910  0.52490  0.5355   
..       ...  ...     ...    ...     ...     ...      ...      ...     ...   
563  0.05623  ...  25.450  26.40  166.10  2027.0  0.14100  0.21130  0.4107   
564  0.05533  ...  23.690  38.25  155.00  1731.0  0.11660  0.19220  0.3215   
565  0.05648  ...  18.980  34.12  126.70  1124.0  0.11390  0.30940  0.3403   
566  0.07016  ...  25.740  39.42  184.60  1821.0  0.16500  0.86810  0.9387   
567  0.05884  ...   9.456  30.37   59.16   268.6  0.08996  0.06444  0.0000   

     0.2654  0.4601   0.1189  
0    0.1860  0.2750  0.08902  
1    0.2430  0.3613  0.08758  
2    0.2575  0.6638  0.17300  
3    0.1625  0.2364  0.07678  
4    0.1741  0.3985  0.12440  
..      ...     ...      ...  
563  0.2216  0.2060  0.07115  
564  0.1628  0.2572  0.06637  
565  0.1418  0.2218  0.07820  
566  0.2650  0.4087  0.12400  
567  0.0000  0.2871  0.07039  

[568 rows x 30 columns]
0      M
1      M
2      M
3      M
4      M
      ..
563    M
564    M
```

```
565    M
566    M
567    B
Name: M, Length: 568, dtype: object
```

```
from sklearn.preprocessing import LabelEncoder

print("Before encoding: ")
print(y[100:110])

labelencoder_Y = LabelEncoder()
y = labelencoder_Y.fit_transform(y)

print("\nAfter encoding: ")
print(y[100:110])
```

```
Before encoding: 
100    B
101    B
102    B
103    B
104    M
105    B
106    B
107    M
108    B
109    B
Name: M, dtype: object

After encoding: 
[0 0 0 0 1 0 0 1 0 0]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from keras.models import Sequential # Used for a plain stack of layers where each layer has exactly one input tensor and one output tensor.
from keras.layers import Dense, Dropout # Dropout randomly sets input units to 0 during training time to avoid overfitting
```

```
# Initialising the ANN
classifier = Sequential()
```

```
classifier.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu', input_dim = 30))
classifier.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
```

```
classifier.fit(X_train, y_train, batch_size = 1, epochs = 250)
```

```
Epoch 1/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6595 - accuracy: 0.6300
Epoch 2/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 3/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6596 - accuracy: 0.6300
Epoch 4/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 5/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6598 - accuracy: 0.6300
Epoch 6/250
454/454 [==============================] - 2s 3ms/step - loss: 0.6596 - accuracy: 0.6300
Epoch 7/250
454/454 [==============================] - 1s 3ms/step - loss: 0.6595 - accuracy: 0.6300
Epoch 8/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 9/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 10/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6595 - accuracy: 0.6300
Epoch 11/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6596 - accuracy: 0.6300
```

```
Epoch 12/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 13/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6596 - accuracy: 0.6300
Epoch 14/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6596 - accuracy: 0.6300
Epoch 15/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6595 - accuracy: 0.6300
Epoch 16/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 17/250
454/454 [==============================] - 1s 3ms/step - loss: 0.6596 - accuracy: 0.6300
Epoch 18/250
454/454 [==============================] - 1s 3ms/step - loss: 0.6596 - accuracy: 0.6300
Epoch 19/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6596 - accuracy: 0.6300
Epoch 20/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 21/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 22/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6595 - accuracy: 0.6300
Epoch 23/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6598 - accuracy: 0.6300
Epoch 24/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 25/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6596 - accuracy: 0.6300
Epoch 26/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6596 - accuracy: 0.6300
Epoch 27/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 28/250
454/454 [==============================] - 1s 2ms/step - loss: 0.6597 - accuracy: 0.6300
Epoch 29/250
454/454 [==============================] - 1s 3ms/step - loss: 0.6597 - accuracy: 0.6300
```

```
from keras.models import load_model
classifier.save('breast_cancer_model.h5') #Save trained ANN
#classifier = load_model('breast_cancer_model.h5')  #Load trained ANN
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
```

```
y_pred = classifier.predict(X_test)
y_pred = [ 1 if y>=0.5 else 0 for y in y_pred ]
```

```
4/4 [==============================] - 0s 3ms/step
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

accuracy = (cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])
print("Accuracy: "+ str(accuracy*100)+"%")
```

```
[[71  0]
 [43  0]]
Accuracy: 62.28070175438597%
```