create a simple neural network (ANN)

```
# import libraries


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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score



# Reading the data.csv file into dataset
df=pd.read_csv("/data.csv")


# number of rows and columns
df.shape
```

```
(569, 33)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
# to see top5 rows
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_r
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1

5 rows × 33 columns

```
del df['Unnamed: 32']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

```
# how many null values w.r.t columns wise
df.isnull().sum()
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
```

```
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

```
# how many null values over all columns
df.isnull().sum().sum()
```

```
0
```

```
# data set into two parts one indepdent and dependet features
X=df.iloc[:, 2:]
y=df.iloc[:, 1]
```

X

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	s
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	

569 rows × 30 columns

y

```
0      M
1      M
2      M
3      M
4      M
      ..
```

```
564    M
565    M
566    M
567    M
568    B
Name: diagnosis, Length: 569, dtype: object
```

```
y.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 569 entries, 0 to 568
Series name: diagnosis
Non-Null Count  Dtype
--------------  -----
569 non-null    object
dtypes: object(1)
memory usage: 4.6+ KB
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   radius_mean              569 non-null    float64
 1   texture_mean             569 non-null    float64
 2   perimeter_mean           569 non-null    float64
 3   area_mean                569 non-null    float64
 4   smoothness_mean          569 non-null    float64
 5   compactness_mean         569 non-null    float64
 6   concavity_mean           569 non-null    float64
 7   concave points_mean      569 non-null    float64
 8   symmetry_mean            569 non-null    float64
 9   fractal_dimension_mean   569 non-null    float64
 10  radius_se                569 non-null    float64
 11  texture_se               569 non-null    float64
 12  perimeter_se             569 non-null    float64
 13  area_se                  569 non-null    float64
 14  smoothness_se            569 non-null    float64
 15  compactness_se           569 non-null    float64
 16  concavity_se             569 non-null    float64
 17  concave points_se        569 non-null    float64
 18  symmetry_se              569 non-null    float64
 19  fractal_dimension_se     569 non-null    float64
 20  radius_worst             569 non-null    float64
 21  texture_worst            569 non-null    float64
 22  perimeter_worst          569 non-null    float64
 23  area_worst               569 non-null    float64
 24  smoothness_worst         569 non-null    float64
 25  compactness_worst        569 non-null    float64
 26  concavity_worst          569 non-null    float64
 27  concave points_worst     569 non-null    float64
 28  symmetry_worst           569 non-null    float64
 29  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder
labelencoder_X_1 = LabelEncoder()
y = labelencoder_X_1.fit_transform(y)
```

```
# Splitting the dataset into the Training set and Test set
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 = 42)

#Feature Scaling
from sklearn.preprocessing import StandardScaler # the values of each columns are different (high and lows are avaialble)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
print(f'the shape of X train is {X_train.shape}')
print(f'the shape of y train is {y_train.shape}')
print(f'the shape of X test is {X_test.shape}')
print(f'the shape of y test is {y_test.shape}')
```

```
     the shape of X train is (455, 30)
     the shape of y train is (455,)
     the shape of X test is (114, 30)
     the shape of y test is (114,)
```

```
#Create  or make the ANN


import keras
from keras.models import Sequential
from keras.layers import Dense,Dropout #sparse or dense
from keras.layers import LeakyReLU,PReLU,ELU # activation functions


# intialize the Empty Artifical Neural Network without inputs and outputs
classifier=Sequential()


# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 16, kernel_initializer = 'he_uniform',activation='relu',input_dim = 30)) #units are output dimensio

classifier.add(Dropout(rate=0.1))



# Adding the second hidden layer
classifier.add(Dense(units=16, kernel_initializer ='he_uniform', activation='relu'))
classifier.add(Dropout(rate=0.1))



# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer='uniform', activation='sigmoid'))
 # for sigmoid we have to use weight intialzers as a glorot



# sigmoid actiavtion function will be used in output layers if your ouptut is binary classification


#compling the ANN
classifier.compile(optimizer='Adamax',loss="binary_crossentropy",metrics=["accuracy"])


# Fitting the ANN to the Training set
annhistory=classifier.fit(X_train, y_train,validation_split=0.22, batch_size=100, epochs=150)
# Long scroll ahead but worth
# The batch size and number of epochs have been set using trial and error. Still looking for more efficient ways and suggestions
```

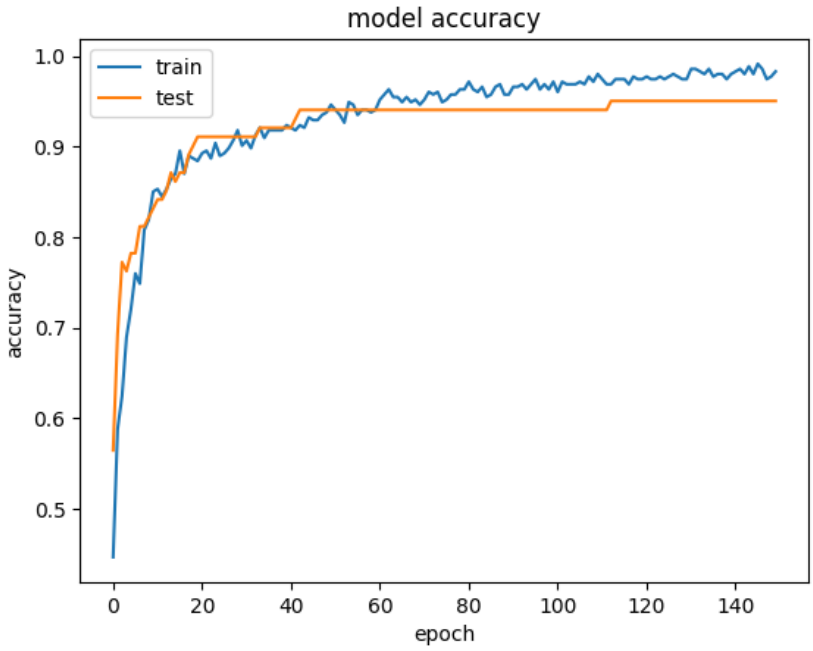
```
Epoch 133/150
4/4 [==============================] - 0s 13ms/step - loss: 0.0697 - accuracy: 0.9831 - val_loss: 0.1498 - val_accuracy:
Epoch 134/150
4/4 [==============================] - 0s 18ms/step - loss: 0.0746 - accuracy: 0.9802 - val_loss: 0.1497 - val_accuracy:
Epoch 135/150
4/4 [==============================] - 0s 13ms/step - loss: 0.0683 - accuracy: 0.9859 - val_loss: 0.1496 - val_accuracy:
Epoch 136/150
4/4 [==============================] - 0s 18ms/step - loss: 0.0713 - accuracy: 0.9774 - val_loss: 0.1494 - val_accuracy:
Epoch 137/150
4/4 [==============================] - 0s 17ms/step - loss: 0.0730 - accuracy: 0.9802 - val_loss: 0.1492 - val_accuracy:
Epoch 138/150
4/4 [==============================] - 0s 17ms/step - loss: 0.0745 - accuracy: 0.9802 - val_loss: 0.1491 - val_accuracy:
Epoch 139/150
4/4 [==============================] - 0s 14ms/step - loss: 0.0763 - accuracy: 0.9746 - val_loss: 0.1490 - val_accuracy:
Epoch 140/150
4/4 [==============================] - 0s 12ms/step - loss: 0.0691 - accuracy: 0.9802 - val_loss: 0.1489 - val_accuracy:
Epoch 141/150
4/4 [==============================] - 0s 12ms/step - loss: 0.0766 - accuracy: 0.9831 - val_loss: 0.1488 - val_accuracy:
Epoch 142/150
4/4 [==============================] - 0s 12ms/step - loss: 0.0680 - accuracy: 0.9859 - val_loss: 0.1486 - val_accuracy:
Epoch 143/150
4/4 [==============================] - 0s 19ms/step - loss: 0.0720 - accuracy: 0.9802 - val_loss: 0.1485 - val_accuracy:
Epoch 144/150
4/4 [==============================] - 0s 17ms/step - loss: 0.0668 - accuracy: 0.9887 - val_loss: 0.1484 - val_accuracy:
Epoch 145/150
4/4 [==============================] - 0s 12ms/step - loss: 0.0735 - accuracy: 0.9802 - val_loss: 0.1484 - val_accuracy:
Epoch 146/150
4/4 [==============================] - 0s 12ms/step - loss: 0.0572 - accuracy: 0.9915 - val_loss: 0.1483 - val_accuracy:
Epoch 147/150
4/4 [==============================] - 0s 20ms/step - loss: 0.0650 - accuracy: 0.9859 - val_loss: 0.1483 - val_accuracy:
Epoch 148/150
4/4 [==============================] - 0s 17ms/step - loss: 0.0734 - accuracy: 0.9746 - val_loss: 0.1482 - val_accuracy:
Epoch 149/150
4/4 [==============================] - 0s 16ms/step - loss: 0.0756 - accuracy: 0.9774 - val_loss: 0.1482 - val_accuracy:
Epoch 150/150
4/4 [==============================] - 0s 20ms/step - loss: 0.0678 - accuracy: 0.9831 - val_loss: 0.1481 - val_accuracy:
```

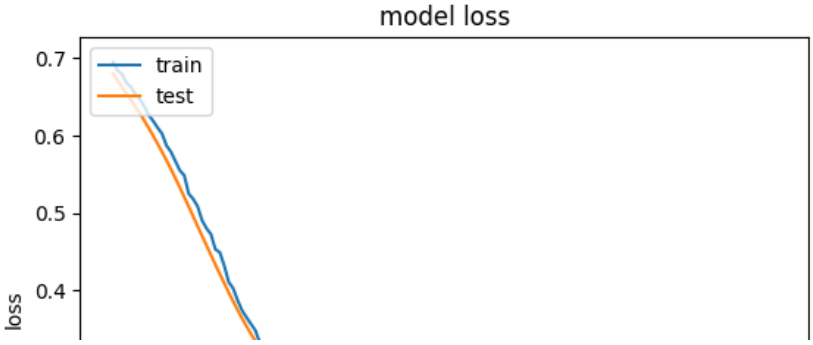
```
print(annhistory.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
#summarize history for accuracy
plt.plot(annhistory.history['accuracy'])
plt.plot(annhistory.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(annhistory.history['loss'])
plt.plot(annhistory.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```






```
# Predictions and Thresholding
y_pred=classifier.predict(X_test)
y_pred=(y_pred>.5)
```

```
4/4 [==============================] - 0s 2ms/step
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
#Accuracy score
from sklearn.metrics import accuracy_score
score=accuracy_score(y_pred,y_test)
```

```
#model Accuracy
score
```

```
0.9824561403508771
```