Importing Library

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder,OrdinalEncoder
from sklearn.metrics import classification_report

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

Loading Dataset

In [ ]:
```
df = pd.read_csv("/content/mushrooms.csv")
df
```

Out[2]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	b	c	l
8120	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	b	v	l
8121	e	f	s	n	f	n	a	c	b	n	...	s	o	o	p	o	o	p	b	c	l
8122	p	k	y	n	f	y	f	c	n	b	...	k	w	w	p	w	o	e	w	v	l
8123	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	o	c	l

8124 rows × 23 columns

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

Getting to know if any column is having a single vlaue

In [ ]:

```
# There is a column called "veil-type" that contains only one category. Although it can be dropped, I have decided to keep it since the dataset is relativel
for col in df.columns:
 print(df[col].value_counts())
 print("------")
```

```
e    4208
p    3916
Name: class, dtype: int64
------
x    3656
f    3152
k     828
b     452
s      32
c       4
Name: cap-shape, dtype: int64
------
y    3244
s    2556
f    2320
g       4
Name: cap-surface, dtype: int64
------
n    2284
g    1840
e    1500
y    1072
w    1040
b     168
p     144
c      44
u      16
r      16
Name: cap-color, dtype: int64
------
f    4748
t    3376
Name: bruises, dtype: int64
------
n    3528
f    2160
y     576
s     576
a     400
l     400
p     256
c     192
m      36
Name: odor, dtype: int64
------
f    7914
a     210
Name: gill-attachment, dtype: int64
------
c    6812
w    1312
Name: gill-spacing, dtype: int64
------
b    5612
n    2512
Name: gill-size, dtype: int64
------
b    1728
p    1492
w    1202
n    1048
g     752
h     732
u     492
k     408
e      96
y      86
o      64
r      24
Name: gill-color, dtype: int64
------
t    4608
e    3516
Name: stalk-shape, dtype: int64
------
b    3776
?    2480
e    1120
c     556
r     192
Name: stalk-root, dtype: int64
------
s    5176
k    2372
f     552
y      24
Name: stalk-surface-above-ring, dtype: int64
------
s    4936
k    2304
f     600
y     284
Name: stalk-surface-below-ring, dtype: int64
------
w    4464
p    1872
g     576
n     448
b     432
o     192
e      96
c      36
y       8
Name: stalk-color-above-ring, dtype: int64
------
w    4384
p    1872
g     576
n     512
b     432
o     192
e      96
c      36
```

```
y      24
Name: stalk-color-below-ring, dtype: int64
------
p    8124
Name: veil-type, dtype: int64
------
w    7924
n      96
o      96
y       8
Name: veil-color, dtype: int64
------
o    7488
t     600
n      36
Name: ring-number, dtype: int64
------
p    3968
e    2776
l    1296
f      48
n      36
Name: ring-type, dtype: int64
------
w    2388
n    1968
k    1872
h    1632
r      72
u      48
o      48
y      48
b      48
Name: spore-print-color, dtype: int64
------
v    4040
y    1712
s    1248
n     400
a     384
c     340
Name: population, dtype: int64
------
d    3148
g    2148
p    1144
l     832
u     368
m     292
w     192
Name: habitat, dtype: int64
------
```

Encoding all the Input features

In [ ]:
```
OE = OrdinalEncoder()
X = OE.fit_transform(df.iloc[:,1:])
X
```

Out[5]:
```
array([[5., 2., 4., ..., 2., 3., 5.],
       [5., 2., 9., ..., 3., 2., 1.],
       [0., 2., 8., ..., 3., 2., 3.],
       ...,
       [2., 2., 4., ..., 0., 1., 2.],
       [3., 3., 4., ..., 7., 4., 2.],
       [5., 2., 4., ..., 4., 1., 2.]])
```

Encoding target feature

In [ ]:
```
LE = LabelEncoder()
Y = LE.fit_transform(df['class'])
Y
```

Out[6]:
```
array([1, 0, 0, ..., 0, 1, 0])
```

Splitting the data into Training Set , Testing Set & Validation Set

In [ ]:
```
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=1)
X_train,X_val,Y_train,Y_val = train_test_split(X_train,Y_train,test_size=0.1,random_state=1)
```

Creating an object of early stopping to save the resources

In [ ]:
```
ES = EarlyStopping(monitor='val_loss',mode='min',patience=8)
```

In [ ]:

```
# obj of Sequential
ann = Sequential()

# adding input/hidden/dropout layers
ann.add(Dense(8,activation='relu'))
ann.add(Dropout(0.6))

ann.add(Dense(6,activation='relu'))
ann.add(Dropout(0.4))

# Output layers
ann.add(Dense(1,activation='sigmoid'))

# compiling the model
ann.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

# fitting the model
ann.fit(X_train, Y_train, batch_size = 15, epochs = 200,callbacks=ES,validation_data=(X_val,Y_val))
```

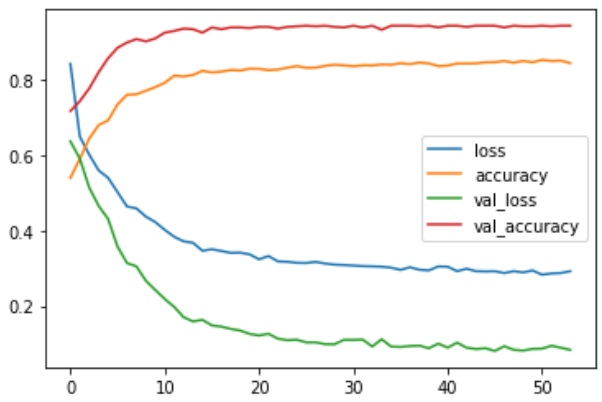
```
Epoch 1/200
390/390 [==============================] - 2s 3ms/step - loss: 0.8419 - accuracy: 0.5396 - val_loss: 0.6366 - val_accuracy: 0.7169
Epoch 2/200
390/390 [==============================] - 1s 2ms/step - loss: 0.6491 - accuracy: 0.5902 - val_loss: 0.5907 - val_accuracy: 0.7431
Epoch 3/200
390/390 [==============================] - 1s 2ms/step - loss: 0.6012 - accuracy: 0.6439 - val_loss: 0.5137 - val_accuracy: 0.7769
Epoch 4/200
390/390 [==============================] - 1s 2ms/step - loss: 0.5595 - accuracy: 0.6798 - val_loss: 0.4654 - val_accuracy: 0.8200
Epoch 5/200
390/390 [==============================] - 1s 3ms/step - loss: 0.5397 - accuracy: 0.6919 - val_loss: 0.4307 - val_accuracy: 0.8569
Epoch 6/200
390/390 [==============================] - 1s 3ms/step - loss: 0.5019 - accuracy: 0.7343 - val_loss: 0.3573 - val_accuracy: 0.8846
Epoch 7/200
390/390 [==============================] - 1s 2ms/step - loss: 0.4638 - accuracy: 0.7600 - val_loss: 0.3138 - val_accuracy: 0.8985
Epoch 8/200
390/390 [==============================] - 1s 3ms/step - loss: 0.4587 - accuracy: 0.7612 - val_loss: 0.3050 - val_accuracy: 0.9077
Epoch 9/200
390/390 [==============================] - 1s 2ms/step - loss: 0.4370 - accuracy: 0.7707 - val_loss: 0.2668 - val_accuracy: 0.9015
Epoch 10/200
390/390 [==============================] - 1s 4ms/step - loss: 0.4223 - accuracy: 0.7803 - val_loss: 0.2423 - val_accuracy: 0.9092
Epoch 11/200
390/390 [==============================] - 1s 4ms/step - loss: 0.4019 - accuracy: 0.7916 - val_loss: 0.2186 - val_accuracy: 0.9246
Epoch 12/200
390/390 [==============================] - 1s 2ms/step - loss: 0.3835 - accuracy: 0.8106 - val_loss: 0.1976 - val_accuracy: 0.9292
Epoch 13/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3713 - accuracy: 0.8085 - val_loss: 0.1704 - val_accuracy: 0.9354
Epoch 14/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3676 - accuracy: 0.8126 - val_loss: 0.1589 - val_accuracy: 0.9338
Epoch 15/200
390/390 [==============================] - 1s 2ms/step - loss: 0.3463 - accuracy: 0.8236 - val_loss: 0.1633 - val_accuracy: 0.9246
Epoch 16/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3504 - accuracy: 0.8193 - val_loss: 0.1484 - val_accuracy: 0.9385
Epoch 17/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3458 - accuracy: 0.8213 - val_loss: 0.1451 - val_accuracy: 0.9338
Epoch 18/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3407 - accuracy: 0.8256 - val_loss: 0.1392 - val_accuracy: 0.9385
Epoch 19/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3413 - accuracy: 0.8244 - val_loss: 0.1349 - val_accuracy: 0.9385
Epoch 20/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3369 - accuracy: 0.8292 - val_loss: 0.1258 - val_accuracy: 0.9369
Epoch 21/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3237 - accuracy: 0.8287 - val_loss: 0.1218 - val_accuracy: 0.9400
Epoch 22/200
390/390 [==============================] - 1s 4ms/step - loss: 0.3323 - accuracy: 0.8253 - val_loss: 0.1262 - val_accuracy: 0.9400
Epoch 23/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3182 - accuracy: 0.8270 - val_loss: 0.1133 - val_accuracy: 0.9354
Epoch 24/200
390/390 [==============================] - 1s 2ms/step - loss: 0.3171 - accuracy: 0.8318 - val_loss: 0.1092 - val_accuracy: 0.9400
Epoch 25/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3144 - accuracy: 0.8359 - val_loss: 0.1104 - val_accuracy: 0.9415
Epoch 26/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3139 - accuracy: 0.8314 - val_loss: 0.1035 - val_accuracy: 0.9431
Epoch 27/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3168 - accuracy: 0.8319 - val_loss: 0.1035 - val_accuracy: 0.9415
Epoch 28/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3126 - accuracy: 0.8366 - val_loss: 0.0987 - val_accuracy: 0.9431
Epoch 29/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3096 - accuracy: 0.8395 - val_loss: 0.0984 - val_accuracy: 0.9400
Epoch 30/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3085 - accuracy: 0.8379 - val_loss: 0.1101 - val_accuracy: 0.9385
Epoch 31/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3071 - accuracy: 0.8357 - val_loss: 0.1100 - val_accuracy: 0.9431
Epoch 32/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3057 - accuracy: 0.8381 - val_loss: 0.1108 - val_accuracy: 0.9385
Epoch 33/200
390/390 [==============================] - 1s 4ms/step - loss: 0.3050 - accuracy: 0.8378 - val_loss: 0.0923 - val_accuracy: 0.9431
Epoch 34/200
390/390 [==============================] - 2s 4ms/step - loss: 0.3042 - accuracy: 0.8403 - val_loss: 0.1117 - val_accuracy: 0.9323
Epoch 35/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3015 - accuracy: 0.8396 - val_loss: 0.0923 - val_accuracy: 0.9431
Epoch 36/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2957 - accuracy: 0.8436 - val_loss: 0.0913 - val_accuracy: 0.9431
Epoch 37/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3026 - accuracy: 0.8412 - val_loss: 0.0937 - val_accuracy: 0.9431
Epoch 38/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2960 - accuracy: 0.8454 - val_loss: 0.0948 - val_accuracy: 0.9415
Epoch 39/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2943 - accuracy: 0.8431 - val_loss: 0.0878 - val_accuracy: 0.9431
Epoch 40/200
390/390 [==============================] - 1s 2ms/step - loss: 0.3048 - accuracy: 0.8360 - val_loss: 0.1001 - val_accuracy: 0.9385
Epoch 41/200
390/390 [==============================] - 1s 3ms/step - loss: 0.3041 - accuracy: 0.8372 - val_loss: 0.0894 - val_accuracy: 0.9431
Epoch 42/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2921 - accuracy: 0.8429 - val_loss: 0.1027 - val_accuracy: 0.9431
Epoch 43/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2987 - accuracy: 0.8427 - val_loss: 0.0893 - val_accuracy: 0.9400
Epoch 44/200
390/390 [==============================] - 1s 4ms/step - loss: 0.2922 - accuracy: 0.8432 - val_loss: 0.0858 - val_accuracy: 0.9431
Epoch 45/200
390/390 [==============================] - 2s 4ms/step - loss: 0.2916 - accuracy: 0.8461 - val_loss: 0.0879 - val_accuracy: 0.9431
Epoch 46/200
390/390 [==============================] - 1s 4ms/step - loss: 0.2921 - accuracy: 0.8463 - val_loss: 0.0808 - val_accuracy: 0.9431
Epoch 47/200
390/390 [==============================] - 1s 4ms/step - loss: 0.2872 - accuracy: 0.8495 - val_loss: 0.0932 - val_accuracy: 0.9385
Epoch 48/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2922 - accuracy: 0.8451 - val_loss: 0.0837 - val_accuracy: 0.9431
Epoch 49/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2889 - accuracy: 0.8497 - val_loss: 0.0817 - val_accuracy: 0.9415
Epoch 50/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2940 - accuracy: 0.8461 - val_loss: 0.0861 - val_accuracy: 0.9415
Epoch 51/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2829 - accuracy: 0.8523 - val_loss: 0.0867 - val_accuracy: 0.9431
Epoch 52/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2862 - accuracy: 0.8495 - val_loss: 0.0944 - val_accuracy: 0.9415
Epoch 53/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2872 - accuracy: 0.8504 - val_loss: 0.0892 - val_accuracy: 0.9431
Epoch 54/200
390/390 [==============================] - 1s 3ms/step - loss: 0.2922 - accuracy: 0.8439 - val_loss: 0.0835 - val_accuracy: 0.9431
```

Out[15]: <keras.callbacks.History at 0x7f552f6722e0>

Visualization of Loss/Accuracy w.r.t Epochs

In [ ]:
```
# as the loss and accuracy goes parallely they will take muchmore time to converge with each other
lossdf = pd.DataFrame(ann.history.history)
lossdf.plot()
```

Out[16]: <AxesSubplot:>



In [ ]:
```
Y_pred = ann.predict(X_test)
```

```
51/51 [==============================] - 0s 2ms/step
```

In [ ]:
```
Y_pred
```

Out[18]:
```
array([[4.36508708e-05],
       [9.79568183e-01],
       [9.98299718e-01],
       ...,
       [1.00000000e+00],
       [1.13329556e-07],
       [1.44340023e-02]], dtype=float32)
```

In [ ]:
```
Y_pred = np.where(Y_pred>0.5,1,0)
```

Classification report

In [ ]:
```
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96       820
           1       1.00      0.91      0.95       805

    accuracy                           0.95      1625
   macro avg       0.96      0.95      0.95      1625
weighted avg       0.96      0.95      0.95      1625
```

In [ ]:
```

```