```
import pandas as pd
```

```
bp = pd.read_csv('/content/backprop.csv')#Import data set
```

```
bp.head()#check the 1st five data
```

```
bp.shape#sahpe of the data
```

```
(8143, 8)
```

```
bp.isnull().sum()#check null value to be remove
```

```
S.No             0
date             0
Temperature      0
Humidity         0
Light            0
CO2              0
HumidityRatio    0
Occupancy        0
dtype: int64
```

```
# Remove unwanted columns
bp1 = bp.drop(['S.No','date'],axis=1)
bp1.head(2)
```

```
bp1.dtypes#data types
```

```
Temperature      float64
Humidity         float64
```

```
Light            float64
CO2              float64
HumidityRatio    float64
Occupancy          int64
dtype: object
```

```
bp1['Occupancy'].unique()#Check unique value of Target variable
```

```
array([1, 0])
```

```
from keras.models import Sequential
from keras.layers import Dense
```

```
X = bp1.drop(['Occupancy'],axis=1).values#.reshape(-1,1)#Independend variable gnerating
y = bp1['Occupancy'].values.reshape(-1,1) #y.values.reshape(-1, 1)#Dependent variable generat
```

```
#split the dataset into testing data (30%) and training data (70%).
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

```
X_train.shape, X_test.shape
```

```
((5700, 5), (2443, 5))
```

Build a model (ANN) in tensorflow/keras

```
import tensorflow as tf
from tensorflow import keras
```

```
model = keras.Sequential([
    keras.layers.Dense(26, input_shape=(X_train.shape[1],), activation='relu'),#Commonly relu
    keras.layers.Dense(15, activation='relu'),#Commonly relu is good for hidden layer
    keras.layers.Dense(1, activation='sigmoid')#Commonly sigmoid is good for output
])
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, epochs=100,batch_size=10,validation_data=(X_test, y_tes
history
```

```
Epoch 73/100
570/570 [==============================] - 2s 4ms/step - loss: 0.0395 - accuracy: 0.9
Epoch 74/100
570/570 [==============================] - 3s 5ms/step - loss: 0.0402 - accuracy: 0.9
Epoch 75/100
570/570 [==============================] - 2s 3ms/step - loss: 0.0395 - accuracy: 0.9
Epoch 76/100
570/570 [==============================] - 1s 2ms/step - loss: 0.0410 - accuracy: 0.9
```

```
570/570 [==============================] - 1s 2ms/step - loss: 0.0410 - accuracy: 0.9
Epoch 77/100
570/570 [==============================] - 1s 2ms/step - loss: 0.0401 - accuracy: 0.9
Epoch 78/100
570/570 [==============================] - 1s 2ms/step - loss: 0.0390 - accuracy: 0.9
Epoch 79/100
570/570 [==============================] - 1s 2ms/step - loss: 0.0389 - accuracy: 0.9
Epoch 80/100
570/570 [==============================] - 1s 2ms/step - loss: 0.0440 - accuracy: 0.9
Epoch 81/100
570/570 [==============================] - 2s 3ms/step - loss: 0.0420 - accuracy: 0.9
Epoch 82/100
570/570 [==============================] - 3s 5ms/step - loss: 0.0412 - accuracy: 0.9
Epoch 83/100
570/570 [==============================] - 4s 6ms/step - loss: 0.0398 - accuracy: 0.9
Epoch 84/100
570/570 [==============================] - 3s 6ms/step - loss: 0.0397 - accuracy: 0.9
Epoch 85/100
570/570 [==============================] - 4s 6ms/step - loss: 0.0414 - accuracy: 0.9
Epoch 86/100
570/570 [==============================] - 2s 4ms/step - loss: 0.0381 - accuracy: 0.9
Epoch 87/100
570/570 [==============================] - 3s 5ms/step - loss: 0.0405 - accuracy: 0.9
Epoch 88/100
570/570 [==============================] - 4s 6ms/step - loss: 0.0419 - accuracy: 0.9
Epoch 89/100
570/570 [==============================] - 4s 8ms/step - loss: 0.0401 - accuracy: 0.9
Epoch 90/100
570/570 [==============================] - 2s 4ms/step - loss: 0.0397 - accuracy: 0.9
Epoch 91/100
570/570 [==============================] - 4s 7ms/step - loss: 0.0408 - accuracy: 0.9
Epoch 92/100
570/570 [==============================] - 3s 6ms/step - loss: 0.0411 - accuracy: 0.9
Epoch 93/100
570/570 [==============================] - 3s 5ms/step - loss: 0.0420 - accuracy: 0.9
Epoch 94/100
570/570 [==============================] - 3s 5ms/step - loss: 0.0426 - accuracy: 0.9
Epoch 95/100
570/570 [==============================] - 2s 4ms/step - loss: 0.0408 - accuracy: 0.9
Epoch 96/100
570/570 [==============================] - 2s 4ms/step - loss: 0.0395 - accuracy: 0.9
Epoch 97/100
570/570 [==============================] - 3s 4ms/step - loss: 0.0389 - accuracy: 0.9
Epoch 98/100
570/570 [==============================] - 2s 4ms/step - loss: 0.0394 - accuracy: 0.9
Epoch 99/100
570/570 [==============================] - 3s 5ms/step - loss: 0.0400 - accuracy: 0.9
Epoch 100/100
570/570 [==============================] - 2s 4ms/step - loss: 0.0401 - accuracy: 0.9
<keras.callbacks.History at 0x7fadd32cd090>
```

```
#What is the highest testing accuracy you were able to achieve from the model?
loss, accuracy=model.evaluate(X_test, y_test)
```

```
print("loss", loss)
print("accuracy", accuracy)#Accuracy means correct predictions
```

```
77/77 [==============================] - 0s 1ms/step - loss: 0.0372 - accuracy: 0.9885
loss 0.03721233457326889
accuracy 0.9885386824607849
```

```
#Let's check our prediction
yp=model.predict(X_test)
yp[:5]
```

```
array([[9.0420789e-01],
       [9.1203117e-01],
       [1.8844018e-10],
       [1.1471204e-11],
       [2.6219001e-12]], dtype=float32)
```

```
#Convert to normal Occupancy
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

```
y_pred[:10]
```

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

```
y_test[:10]
```

```
array([[0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [0]])
```

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

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1937
           1       0.95      1.00      0.97       506

    accuracy                           0.99      2443
```

```
   macro avg       0.97      0.99      0.98      2443
weighted avg       0.99      0.99      0.99      2443
```

accuracy - 99%

```
import seaborn as sn
from matplotlib import pyplot as plt
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_pred)

plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
history_dict = history.history
print(history_dict.keys())
```

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

```
#accuracy = history_dict['accuracy']
#val_accuracy = history_dict['val_acccuracy']
```

```
#Plot the training accuracy and the testing accuracy with the no of epochs in one plot
```

```
trainloss = history.history['accuracy']
valid_loss = history.history['val_accuracy']
epochs = range(1,101)
plt.plot(epochs, trainloss, 'g', label='Training accuracy')
plt.plot(epochs, valid_loss, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('# of Epochs')
plt.ylabel('Accuracy %')
plt.legend()
plt.show()
```

✓ 1s completed at 10:16 AM

✓ 1s completed at 10:10 AM