If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

* Zero configuration required
* Access to GPUs free of charge
* Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](https://www.youtube.com/watch?v=inN8seMm7UI) to learn more, or just get started below!

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

seconds\_in\_a\_day = 24 \* 60 \* 60  
seconds\_in\_a\_day

86400

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

seconds\_in\_a\_week = 7 \* seconds\_in\_a\_day  
seconds\_in\_a\_week

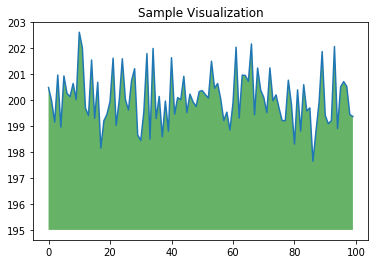
604800

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see [Overview of Colab](/notebooks/basic_features_overview.ipynb). To create a new Colab notebook you can use the File menu above, or use the following link: [create a new Colab notebook](http://colab.research.google.com#create=true).

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see [jupyter.org](https://www.jupyter.org).

With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

import numpy as np  
from matplotlib import pyplot as plt  
  
ys = 200 + np.random.randn(100)  
x = [x for x in range(len(ys))]  
  
plt.plot(x, ys, '-')  
plt.fill\_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)  
  
plt.title("Sample Visualization")  
plt.show()



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under [Working with Data](#working-with-data).

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just [a few lines of code](https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quickstart/beginner.ipynb). Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including [GPUs and TPUs](#using-accelerated-hardware), regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including:

* Getting started with TensorFlow
* Developing and training neural networks
* Experimenting with TPUs
* Disseminating AI research
* Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the [machine learning examples](#machine-learning-examples) below.

* [Overview of Colaboratory](/notebooks/basic_features_overview.ipynb)
* [Guide to Markdown](/notebooks/markdown_guide.ipynb)
* [Importing libraries and installing dependencies](/notebooks/snippets/importing_libraries.ipynb)
* [Saving and loading notebooks in GitHub](https://colab.research.google.com/github/googlecolab/colabtools/blob/main/notebooks/colab-github-demo.ipynb)
* [Interactive forms](/notebooks/forms.ipynb)
* [Interactive widgets](/notebooks/widgets.ipynb)
* [Loading data: Drive, Sheets, and Google Cloud Storage](/notebooks/io.ipynb)
* [Charts: visualizing data](/notebooks/charts.ipynb)
* [Getting started with BigQuery](/notebooks/bigquery.ipynb)

### Machine Learning Crash Course

These are a few of the notebooks from Google's online Machine Learning course. See the [full course website](https://developers.google.com/machine-learning/crash-course/) for more.

* [Intro to Pandas DataFrame](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/pandas_dataframe_ultraquick_tutorial.ipynb)
* [Linear regression with tf.keras using synthetic data](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/linear_regression_with_synthetic_data.ipynb)
* [TensorFlow with GPUs](/notebooks/gpu.ipynb)
* [TensorFlow with TPUs](/notebooks/tpu.ipynb)
* [NeMo Voice Swap](https://colab.research.google.com/github/NVIDIA/NeMo/blob/stable/tutorials/VoiceSwapSample.ipynb): Use Nvidia's NeMo conversational AI Toolkit to swap a voice in an audio fragment with a computer generated one.
* [Retraining an Image Classifier](https://tensorflow.org/hub/tutorials/tf2_image_retraining): Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
* [Text Classification](https://tensorflow.org/hub/tutorials/tf2_text_classification): Classify IMDB movie reviews as either *positive* or *negative*.
* [Style Transfer](https://tensorflow.org/hub/tutorials/tf2_arbitrary_image_stylization): Use deep learning to transfer style between images.
* [Multilingual Universal Sentence Encoder Q&A](https://tensorflow.org/hub/tutorials/retrieval_with_tf_hub_universal_encoder_qa): Use a machine learning model to answer questions from the SQuAD dataset.
* [Video Interpolation](https://tensorflow.org/hub/tutorials/tweening_conv3d): Predict what happened in a video between the first and the last frame.

#import required libraries  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns

df=pd.read\_csv('/content/drug200.csv')  
df

Age Sex BP Cholesterol Na\_to\_K Drug  
0 23 F HIGH HIGH 25.355 DrugY  
1 47 M LOW HIGH 13.093 drugC  
2 47 M LOW HIGH 10.114 drugC  
3 28 F NORMAL HIGH 7.798 drugX  
4 61 F LOW HIGH 18.043 DrugY  
.. ... .. ... ... ... ...  
195 56 F LOW HIGH 11.567 drugC  
196 16 M LOW HIGH 12.006 drugC  
197 52 M NORMAL HIGH 9.894 drugX  
198 23 M NORMAL NORMAL 14.020 drugX  
199 40 F LOW NORMAL 11.349 drugX  
  
[200 rows x 6 columns]

df.head()

Age Sex BP Cholesterol Na\_to\_K Drug  
0 23 F HIGH HIGH 25.355 DrugY  
1 47 M LOW HIGH 13.093 drugC  
2 47 M LOW HIGH 10.114 drugC  
3 28 F NORMAL HIGH 7.798 drugX  
4 61 F LOW HIGH 18.043 DrugY

df.tail()

Age Sex BP Cholesterol Na\_to\_K Drug  
195 56 F LOW HIGH 11.567 drugC  
196 16 M LOW HIGH 12.006 drugC  
197 52 M NORMAL HIGH 9.894 drugX  
198 23 M NORMAL NORMAL 14.020 drugX  
199 40 F LOW NORMAL 11.349 drugX

CHECKING NULL VALUES

df.isnull().sum()

Age 0  
Sex 0  
BP 0  
Cholesterol 0  
Na\_to\_K 0  
Drug 0  
dtype: int64

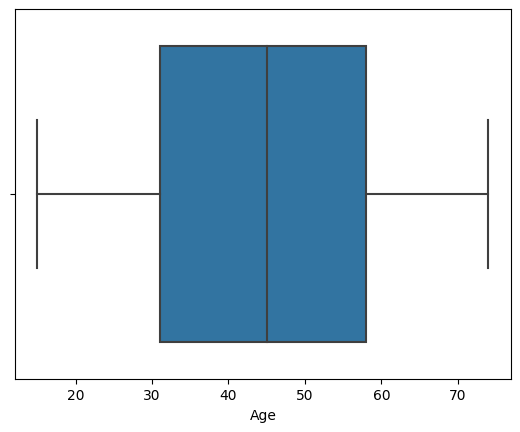
df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 200 entries, 0 to 199  
Data columns (total 6 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Age 200 non-null int64   
 1 Sex 200 non-null object   
 2 BP 200 non-null object   
 3 Cholesterol 200 non-null object   
 4 Na\_to\_K 200 non-null float64  
 5 Drug 200 non-null object   
dtypes: float64(1), int64(1), object(4)  
memory usage: 9.5+ KB

FINDING OUTLIERS

sns.boxplot(x=df.Age)

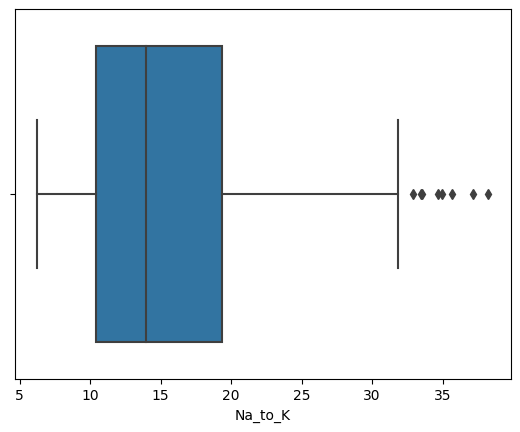
<Axes: xlabel='Age'>



for age attribute there is no outliers so no need replace the values

sns.boxplot(x=df.Na\_to\_K)

<Axes: xlabel='Na\_to\_K'>



REPLACING THE OUTLIERS USING MEDAIN METHOD

q1 = df.Na\_to\_K.quantile(.25)  
q3 = df.Na\_to\_K.quantile(.75)

IQR = q3 - q1  
IQR

8.9345

upper\_limit = q3+1.5\*IQR  
lower\_limit = q1-1.5\*IQR

df.median()

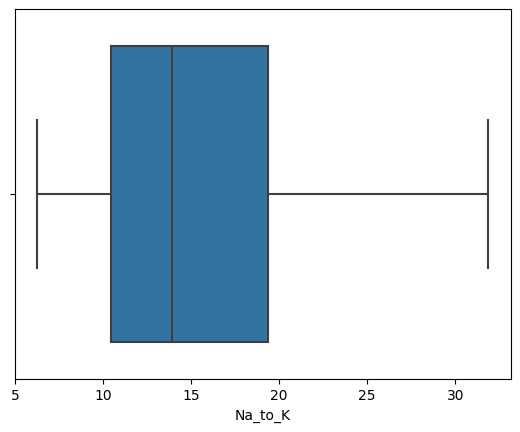
<ipython-input-18-6d467abf240d>:1: FutureWarning: The default value of numeric\_only in DataFrame.median is deprecated. In a future version, it will default to False. In addition, specifying 'numeric\_only=None' is deprecated. Select only valid columns or specify the value of numeric\_only to silence this warning.  
 df.median()

Age 45.0000  
Na\_to\_K 13.9365  
dtype: float64

df['Na\_to\_K'] =np.where(df['Na\_to\_K']>upper\_limit,30.4,df['Na\_to\_K'])

sns.boxplot(x=df.Na\_to\_K)

<Axes: xlabel='Na\_to\_K'>



THERE IS NO OUTLIERS

we have splitting the data step:1-->split dependent and independent(target column is dependent others are independent) step:2-->split training and testing data

x=df.loc[:,'Age':'Na\_to\_K']   
x.head()

Age Sex BP Cholesterol Na\_to\_K  
0 23 F HIGH HIGH 25.355  
1 47 M LOW HIGH 13.093  
2 47 M LOW HIGH 10.114  
3 28 F NORMAL HIGH 7.798  
4 61 F LOW HIGH 18.043

y=df['Drug'] #dependent variable  
y.head()  
y=pd.DataFrame(y)  
y

Drug  
0 DrugY  
1 drugC  
2 drugC  
3 drugX  
4 DrugY  
.. ...  
195 drugC  
196 drugC  
197 drugX  
198 drugX  
199 drugX  
  
[200 rows x 1 columns]

#after spliting the dependent and independent data we have to split the training and testing data

* List item
* List item

from sklearn.model\_selection import train\_test\_split

#after spliting the dependent and independent data we have to split the training and testing data  
xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.2,random\_state=21) #0.2-->20%

xtrain.shape,xtest.shape

((160, 5), (40, 5))

xtrain

Age Sex BP Cholesterol Na\_to\_K  
176 48 M HIGH NORMAL 10.446  
111 47 F NORMAL NORMAL 6.683  
114 20 F NORMAL NORMAL 9.281  
14 50 F NORMAL HIGH 12.703  
106 22 M NORMAL HIGH 11.953  
.. ... .. ... ... ...  
120 28 M NORMAL HIGH 27.064  
112 35 M LOW NORMAL 9.170  
48 23 M NORMAL HIGH 31.686  
4 61 F LOW HIGH 18.043  
56 65 M HIGH NORMAL 11.340  
  
[160 rows x 5 columns]

xtest

Age Sex BP Cholesterol Na\_to\_K  
144 39 M HIGH HIGH 9.664  
9 43 M LOW NORMAL 19.368  
17 43 M HIGH HIGH 13.972  
20 57 M LOW NORMAL 19.128  
45 66 F NORMAL NORMAL 8.107  
6 49 F NORMAL HIGH 16.275  
52 62 M LOW NORMAL 27.183  
91 41 M HIGH NORMAL 15.156  
129 32 F NORMAL HIGH 7.477  
183 36 F HIGH NORMAL 15.490  
21 63 M NORMAL HIGH 25.917  
42 50 M NORMAL NORMAL 15.790  
105 37 M LOW NORMAL 8.968  
145 61 M NORMAL HIGH 9.443  
87 69 M LOW HIGH 15.478  
152 55 M NORMAL NORMAL 7.261  
124 53 F HIGH NORMAL 12.495  
92 29 F HIGH HIGH 29.450  
101 45 F HIGH HIGH 12.854  
13 74 F LOW HIGH 20.942  
108 72 M HIGH NORMAL 9.677  
75 26 M LOW NORMAL 20.909  
40 73 F NORMAL HIGH 19.221  
198 23 M NORMAL NORMAL 14.020  
191 23 M HIGH HIGH 8.011  
192 72 M LOW HIGH 16.310  
194 46 F HIGH HIGH 30.400  
117 40 F NORMAL HIGH 10.103  
2 47 M LOW HIGH 10.114  
131 52 M LOW NORMAL 30.400  
19 32 F HIGH NORMAL 25.974  
143 74 M HIGH NORMAL 15.436  
147 26 F HIGH NORMAL 12.307  
22 47 M LOW NORMAL 30.568  
102 28 F LOW HIGH 13.127  
60 38 F LOW NORMAL 29.875  
136 55 F HIGH HIGH 10.977  
168 51 F LOW NORMAL 23.003  
138 51 M HIGH NORMAL 11.343  
54 68 F HIGH NORMAL 10.189

ytrain.shape,ytest.shape

((160, 1), (40, 1))

ytrain

Drug  
176 drugA  
111 drugX  
114 drugX  
14 drugX  
106 drugX  
.. ...  
120 DrugY  
112 drugX  
48 DrugY  
4 DrugY  
56 drugB  
  
[160 rows x 1 columns]

ytest

Drug  
144 drugA  
9 DrugY  
17 drugA  
20 DrugY  
45 drugX  
6 DrugY  
52 DrugY  
91 DrugY  
129 drugX  
183 DrugY  
21 DrugY  
42 DrugY  
105 drugX  
145 drugX  
87 DrugY  
152 drugX  
124 drugB  
92 DrugY  
101 drugA  
13 DrugY  
108 drugB  
75 DrugY  
40 DrugY  
198 drugX  
191 drugA  
192 DrugY  
194 DrugY  
117 drugX  
2 drugC  
131 DrugY  
19 DrugY  
143 DrugY  
147 drugA  
22 DrugY  
102 drugC  
60 DrugY  
136 drugB  
168 DrugY  
138 drugB  
54 drugB

whenever we built the model the datatype should be in int or float otherwise it will give error Encoding--> one hot encoding,label encoding,manual encoding

#import libraries for the encoding (label encoding)  
from sklearn.preprocessing import LabelEncoder

#initialise the library  
le=LabelEncoder()

xtrain['Sex'] = le.fit\_transform(xtrain['Sex'])  
xtrain['Sex']

176 1  
111 0  
114 0  
14 0  
106 1  
 ..  
120 1  
112 1  
48 1  
4 0  
56 1  
Name: Sex, Length: 160, dtype: int64

xtest['Sex'] = le.transform(xtest['Sex'])  
xtest['Sex']

144 1  
9 1  
17 1  
20 1  
45 0  
6 0  
52 1  
91 1  
129 0  
183 0  
21 1  
42 1  
105 1  
145 1  
87 1  
152 1  
124 0  
92 0  
101 0  
13 0  
108 1  
75 1  
40 0  
198 1  
191 1  
192 1  
194 0  
117 0  
2 1  
131 1  
19 0  
143 1  
147 0  
22 1  
102 0  
60 0  
136 0  
168 0  
138 1  
54 0  
Name: Sex, dtype: int64

xtrain['BP'] = le.fit\_transform(xtrain['BP'])  
xtrain['BP']

176 0  
111 2  
114 2  
14 2  
106 2  
 ..  
120 2  
112 1  
48 2  
4 1  
56 0  
Name: BP, Length: 160, dtype: int64

xtest['BP'] = le.transform(xtest['BP'])  
xtest['BP']

144 0  
9 1  
17 0  
20 1  
45 2  
6 2  
52 1  
91 0  
129 2  
183 0  
21 2  
42 2  
105 1  
145 2  
87 1  
152 2  
124 0  
92 0  
101 0  
13 1  
108 0  
75 1  
40 2  
198 2  
191 0  
192 1  
194 0  
117 2  
2 1  
131 1  
19 0  
143 0  
147 0  
22 1  
102 1  
60 1  
136 0  
168 1  
138 0  
54 0  
Name: BP, dtype: int64

xtrain['Cholesterol'] = le.fit\_transform(xtrain['Cholesterol'])  
xtrain['Cholesterol']

176 1  
111 1  
114 1  
14 0  
106 0  
 ..  
120 0  
112 1  
48 0  
4 0  
56 1  
Name: Cholesterol, Length: 160, dtype: int64

xtest['Cholesterol'] = le.transform(xtest['Cholesterol'])  
xtest['Cholesterol']

144 0  
9 1  
17 0  
20 1  
45 1  
6 0  
52 1  
91 1  
129 0  
183 1  
21 0  
42 1  
105 1  
145 0  
87 0  
152 1  
124 1  
92 0  
101 0  
13 0  
108 1  
75 1  
40 0  
198 1  
191 0  
192 0  
194 0  
117 0  
2 0  
131 1  
19 1  
143 1  
147 1  
22 1  
102 0  
60 1  
136 0  
168 1  
138 1  
54 1  
Name: Cholesterol, dtype: int64

ytrain['Drug'] = le.fit\_transform(ytrain['Drug'])  
ytrain['Drug']

176 1  
111 4  
114 4  
14 4  
106 4  
 ..  
120 0  
112 4  
48 0  
4 0  
56 2  
Name: Drug, Length: 160, dtype: int64

ytest['Drug'] = le.transform(ytest['Drug'])  
ytest['Drug']

144 1  
9 0  
17 1  
20 0  
45 4  
6 0  
52 0  
91 0  
129 4  
183 0  
21 0  
42 0  
105 4  
145 4  
87 0  
152 4  
124 2  
92 0  
101 1  
13 0  
108 2  
75 0  
40 0  
198 4  
191 1  
192 0  
194 0  
117 4  
2 3  
131 0  
19 0  
143 0  
147 1  
22 0  
102 3  
60 0  
136 2  
168 0  
138 2  
54 2  
Name: Drug, dtype: int64

#after encoding the data  
xtrain

Age Sex BP Cholesterol Na\_to\_K  
176 48 1 0 1 10.446  
111 47 0 2 1 6.683  
114 20 0 2 1 9.281  
14 50 0 2 0 12.703  
106 22 1 2 0 11.953  
.. ... ... .. ... ...  
120 28 1 2 0 27.064  
112 35 1 1 1 9.170  
48 23 1 2 0 31.686  
4 61 0 1 0 18.043  
56 65 1 0 1 11.340  
  
[160 rows x 5 columns]

xtest

Age Sex BP Cholesterol Na\_to\_K  
144 39 1 0 0 9.664  
9 43 1 1 1 19.368  
17 43 1 0 0 13.972  
20 57 1 1 1 19.128  
45 66 0 2 1 8.107  
6 49 0 2 0 16.275  
52 62 1 1 1 27.183  
91 41 1 0 1 15.156  
129 32 0 2 0 7.477  
183 36 0 0 1 15.490  
21 63 1 2 0 25.917  
42 50 1 2 1 15.790  
105 37 1 1 1 8.968  
145 61 1 2 0 9.443  
87 69 1 1 0 15.478  
152 55 1 2 1 7.261  
124 53 0 0 1 12.495  
92 29 0 0 0 29.450  
101 45 0 0 0 12.854  
13 74 0 1 0 20.942  
108 72 1 0 1 9.677  
75 26 1 1 1 20.909  
40 73 0 2 0 19.221  
198 23 1 2 1 14.020  
191 23 1 0 0 8.011  
192 72 1 1 0 16.310  
194 46 0 0 0 30.400  
117 40 0 2 0 10.103  
2 47 1 1 0 10.114  
131 52 1 1 1 30.400  
19 32 0 0 1 25.974  
143 74 1 0 1 15.436  
147 26 0 0 1 12.307  
22 47 1 1 1 30.568  
102 28 0 1 0 13.127  
60 38 0 1 1 29.875  
136 55 0 0 0 10.977  
168 51 0 1 1 23.003  
138 51 1 0 1 11.343  
54 68 0 0 1 10.189

ytrain

Drug  
176 1  
111 4  
114 4  
14 4  
106 4  
.. ...  
120 0  
112 4  
48 0  
4 0  
56 2  
  
[160 rows x 1 columns]

PREPROCESSING IS DONE THE DATA IS READY TO COMPILE

TASK 2: DEVELOP THE MODEL

#BUILD AN ANN MODEL  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense

# Initializing the seq model  
reg\_model = Sequential()  
# Adding the input layer to the model  
reg\_model.add(Dense(4,activation='relu'))  
# Adding the 1st hidden layer to the model  
reg\_model.add(Dense(64,activation='relu'))  
# Adding the 2nd hidden layer to the model  
reg\_model.add(Dense(32,activation='relu'))  
# Adding the 2nd hidden layer to the model  
reg\_model.add(Dense(16,activation='relu'))  
# Adding output layer  
reg\_model.add(Dense(1,activation='softmax'))

#compile the model  
reg\_model.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=['accuracy'])

#train and test the model with the help of(xtrain,xtest,ytrain,ytest)  
reg\_model.fit(xtrain,ytrain,batch\_size=4,epochs=15,validation\_data=(xtest,ytest))

Epoch 1/15

/usr/local/lib/python3.10/dist-packages/tensorflow/python/util/dispatch.py:1176: SyntaxWarning: In loss categorical\_crossentropy, expected y\_pred.shape to be (batch\_size, num\_classes) with num\_classes > 1. Received: y\_pred.shape=(4, 1). Consider using 'binary\_crossentropy' if you only have 2 classes.  
 return dispatch\_target(\*args, \*\*kwargs)

40/40 [==============================] - 2s 16ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 2/15  
31/40 [======================>.......] - ETA: 0s - loss: 0.0000e+00 - accuracy: 0.1129

/usr/local/lib/python3.10/dist-packages/tensorflow/python/util/dispatch.py:1176: SyntaxWarning: In loss categorical\_crossentropy, expected y\_pred.shape to be (batch\_size, num\_classes) with num\_classes > 1. Received: y\_pred.shape=(4, 1). Consider using 'binary\_crossentropy' if you only have 2 classes.  
 return dispatch\_target(\*args, \*\*kwargs)

40/40 [==============================] - 0s 3ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 3/15  
40/40 [==============================] - 0s 3ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 4/15  
40/40 [==============================] - 0s 3ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 5/15  
40/40 [==============================] - 0s 3ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 6/15  
40/40 [==============================] - 0s 3ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 7/15  
40/40 [==============================] - 0s 3ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 8/15  
40/40 [==============================] - 0s 3ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 9/15  
40/40 [==============================] - 0s 6ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 10/15  
40/40 [==============================] - 0s 5ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 11/15  
40/40 [==============================] - 0s 5ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 12/15  
40/40 [==============================] - 0s 4ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 13/15  
40/40 [==============================] - 0s 4ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 14/15  
40/40 [==============================] - 0s 4ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250  
Epoch 15/15  
40/40 [==============================] - 0s 4ms/step - loss: 0.0000e+00 - accuracy: 0.1125 - val\_loss: 0.0000e+00 - val\_accuracy: 0.1250

<keras.callbacks.History at 0x7ff141dc5120>

TASK 3: Test the model with random data

#Test with random data  
out=reg\_model.predict([[345,345,3455,2,500]])  
out

1/1 [==============================] - 0s 165ms/step

array([[1.]], dtype=float32)