Step_1 : importing data and unzipping

```
!wget https://upscfever.com/datasets/flowers-new.zip
```

```
--2023-01-24 15:38:22--  https://upscfever.com/datasets/flowers-new.zip
Resolving upscfever.com (upscfever.com)... 172.67.193.2, 104.21.90.10, 2606:4700:3033::6815:5a0a, ...
Connecting to upscfever.com (upscfever.com)|172.67.193.2|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: ‘flowers-new.zip’

flowers-new.zip         [    <=>              ]   5.43M   351KB/s    in 12s

2023-01-24 15:38:35 (470 KB/s) - Read error at byte 5696234 (Success).Retrying.

--2023-01-24 15:38:36--  (try: 2)  https://upscfever.com/datasets/flowers-new.zip
Connecting to upscfever.com (upscfever.com)|172.67.193.2|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: ‘flowers-new.zip’

flowers-new.zip         [            <=>      ]   5.74M  2.00MB/s    in 2.9s

2023-01-24 15:38:40 (2.00 MB/s) - ‘flowers-new.zip’ saved [6021364]
```

```
!unzip flowers-new.zip
```

```
Archive:  flowers-new.zip
   creating: flowers/
   creating: flowers/daisy/
  inflating: flowers/daisy/100080576_f52e8ee070_n.jpg
  inflating: flowers/daisy/11642632_1e7627a2cc.jpg
  inflating: flowers/daisy/15207766_fc2f1d692c_n.jpg
  inflating: flowers/daisy/21652746_cc379e0eea_m.jpg
  inflating: flowers/daisy/25360380_1a881a5648.jpg
  inflating: flowers/daisy/43474673_7bb4465a86.jpg
  inflating: flowers/daisy/54377391_15648e8d18.jpg
  inflating: flowers/daisy/5547758_eea9edfd54_n.jpg
  inflating: flowers/daisy/5673551_01d1ea993e_n.jpg
  inflating: flowers/daisy/5673728_71b8cb57eb.jpg
  inflating: flowers/daisy/5794835_d15905c7c8_n.jpg
  inflating: flowers/daisy/5794839_200acd910c_n.jpg
  inflating: flowers/daisy/99306615_739eb94b9e_m.jpg
   creating: flowers/dandelion/
  inflating: flowers/dandelion/10443973_aeb97513fc_m.jpg
  inflating: flowers/dandelion/10683189_bd6e371b97.jpg
  inflating: flowers/dandelion/10919961_0af657c4e8.jpg
  inflating: flowers/dandelion/11405573_24a8a838cc_n.jpg
  inflating: flowers/dandelion/11545123_50a340b473_m.jpg
  inflating: flowers/dandelion/126012913_edf771c564_n.jpg
  inflating: flowers/dandelion/13290033_ebd7c7abba_n.jpg
  inflating: flowers/dandelion/13920113_f03e867ea7_m.jpg
  inflating: flowers/dandelion/14283011_3e7452c5b2_n.jpg
  inflating: flowers/dandelion/14829055_2a2e646a8f_m.jpg
  inflating: flowers/dandelion/15987457_49dc11bf4b.jpg
```

```
    inflating: flowers/dandelion/16041975_2f6c1596e5.jpg
    inflating: flowers/dandelion/16159487_3a6615a565_n.jpg
    inflating: flowers/dandelion/16987075_9a690a2183.jpg
    inflating: flowers/dandelion/61242541_a04395e6bc.jpg
    inflating: flowers/dandelion/62293290_2c463891ff_m.jpg
    inflating: flowers/dandelion/7355522_b66e5d3078_m.jpg
    inflating: flowers/dandelion/80846315_d997645bea_n.jpg
    inflating: flowers/dandelion/8181477_8cb77d2e0f_n.jpg
    inflating: flowers/dandelion/8223949_2928d3f6f6_n.jpg
    inflating: flowers/dandelion/8223968_6b51555d2f_n.jpg
    inflating: flowers/dandelion/8475758_4c861ab268_m.jpg
    inflating: flowers/dandelion/8475769_3dea463364_m.jpg
    inflating: flowers/dandelion/8684108_a85764b22d_n.jpg
    inflating: flowers/dandelion/9818247_e2eac18894.jpg
    inflating: flowers/dandelion/98992760_53ed1d26a9.jpg
     creating: flowers/rose/
    inflating: flowers/rose/102501987_3cdb8e5394_n.jpg
    inflating: flowers/rose/110472418_87b6a3aa98_m.jpg
    inflating: flowers/rose/118974357_0faa23cce9_n.jpg
    inflating: flowers/rose/12240303_80d87f77a3_n.jpg
    inflating: flowers/rose/123128873_546b8b7355_n.jpg
    inflating: flowers/rose/145862135_ab710de93c_n.jpg
    inflating: flowers/rose/159079265_d77a9ac920_n.jpg
    inflating: flowers/rose/160954292_6c2b4fda65_n.jpg
    inflating: flowers/rose/172311368_49412f881b.jpg
    inflating: flowers/rose/174109630_3c544b8a2f.jpg
    inflating: flowers/rose/180613732_3a7aba0b80_n.jpg
    inflating: flowers/rose/218630974_5646dafc63_m.jpg
    inflating: flowers/rose/22679076_bdb4c24401_m.jpg
    inflating: flowers/rose/229488796_21ac6ee16d_n.jpg
```

```
#jupyter-> from zipfile import ZipFile
 #          ZipFile("flowers-new.zip").extractall("C:/Users/admin")
```

Step_2 : importing libraries

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2        #jupyter-> pip install opencv-python   #cv2 is used for reading images


X = []
y = []


                   #step_2.1 : Reading jpg files( also adding values to above given X and y)
'''
reading jpg files to X, y
'''
def readingFiles(folderpath,foldername):

  paths=os.path.join(folderpath, foldername)

  for img in os.listdir(paths):#os.listdir - get names of files inside a folder given the path of folder
```

```
   #read first file name
   filepath = os.path.join(paths, img)
   #read image
   img_read = cv2.imread(filepath, cv2.IMREAD_COLOR)
   #resize image
   img_resized = cv2.resize(img_read, (256,256))
   #covert to numpy
   img_np = np.array(img_resized)
   #add to X
   X.append(img_np)
   #add foldername as label to y
   y.append(foldername)
```

```
folderpath= '/content/flowers'

flowername = os.listdir('/content/flowers')

for i in flowername:
 readingFiles(folderpath, i)
```

```
X       #what this output is indicating
```

```
     [array([[[ 54, 130,  96],
              [ 50, 124,  87],
              [ 48, 119,  79],
              ...,
              [ 14,  63,  30],
              [ 14,  65,  31],
              [ 17,  68,  34]],

             [[ 65, 146, 105],
              [ 60, 139,  97],
              [ 58, 133,  90],
              ...,
              [ 15,  64,  32],
              [ 15,  65,  32],
              [ 15,  68,  34]],

             [[ 70, 157, 110],
              [ 65, 149, 101],
              [ 62, 144,  94],
              ...,
              [ 16,  66,  34],
              [ 15,  66,  34],
              [ 16,  68,  35]],

             ...,

             [[250, 250, 250],
              [251, 251, 251],
              [251, 251, 251],
              ...,
              [  9,  27,  14],
              [ 11,  26,  15],
              [ 11,  24,  15]],
```

```
        [[249, 249, 249],
         [250, 250, 250],
         [247, 247, 247],
         ...,
         [  9,  29,  15],
         [ 11,  29,  15],
         [ 12,  27,  15]],

        [[251, 251, 251],
         [251, 251, 251],
         [244, 244, 244],
         ...,
         [ 11,  31,  16],
         [ 13,  31,  18],
         [ 13,  30,  17]]], dtype=uint8), array([[[146, 163, 160],
         [149, 166, 163],
         [151, 166, 164],
         ...,
         [194, 194, 191],
         [194, 192, 186],
         [201, 196, 188]],

        [[145, 162, 159],
         [148, 165, 162],
```

```
len(X)
```

```
117
```

```
y
```

```
['daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'daisy',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
 'dandelion',
```

```
       'dandelion',
       'dandelion',
       'dandelion',
       'dandelion',
       'dandelion',
       'dandelion',
       'dandelion',
       'dandelion',
       'dandelion',
       'dandelion',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
       'tulip',
```

```
len(y)
```

```
117
```

```
               #Step_2.2 : converting X and y to numpy arrays

X = np.array(X)
y = np.array(y)
```

Step_3 : one hot encoding

step_3.1 : LableEncoding and ohe of y (because it contains strings)

```
from sklearn.preprocessing import LabelEncoder #as y is having string values(name of types of flowers); in order to convert them into label ..we use this
from tensorflow.keras.utils import to_categorical #afterwards labels are then converted into classes using this code that is to_categorical()
```

```
#labeling flower names inside flower folder

enc = LabelEncoder()
y_le = enc.fit_transform(y)
```

```
y_le
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3])
```

```
#ohe

y_ohe = to_categorical(y_le, num_classes=5)
```

```
y_ohe
```

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

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

step_3.2 : Standardisation of X

```
X_scaled = X/255
```

Step_4 : splittting data

```
'''
test_size=0.2, [20pc size of test set]
random_state=34, [generating randomly numbers]
shuffle=True, [shuffling dataset]
stratify=y_np [stratified sampling]
'''
```

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_scaled, y_ohe, test_size=0.2, random_state=34, shuffle=True, stratify=y_ohe)

Xtrain, Xval, ytrain, yval = train_test_split(Xtrain, ytrain, test_size=0.2, random_state=34, shuffle=True, stratify=ytrain)
```

```
Xtrain.shape
```

```
(74, 256, 256, 3)
```

```
Xval.shape
```

```
(19, 256, 256, 3)
```

```
Xtest
```

```
array([[[[0.00392157, 0.00392157, 0.00392157],
         [0.00392157, 0.00392157, 0.00392157],
         [0.00392157, 0.00392157, 0.00392157],
         ...,
         [0.00392157, 0.00392157, 0.00392157],
         [0.00392157, 0.00392157, 0.00392157],
         [0.00392157, 0.00392157, 0.00392157]],
```

```
[[0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 ...,
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157]],

[[0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 ...,
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157]],

...,

[[0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 ...,
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157]],

[[0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 ...,
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157]],

[[0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 ...,
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157],
 [0.00392157, 0.00392157, 0.00392157]]],


[[[0.45882353, 0.25490196, 0.16470588],
 [0.45882353, 0.25490196, 0.16470588],
 [0.45490196, 0.25098039, 0.16078431],
 ...,
 [0.40784314, 0.23529412, 0.16470588],
 [0.40392157, 0.22352941, 0.15686275],
```

```
Xtest.shape
```

```
(24, 256, 256, 3)
```

Step_5 : ANN

```
'''
Note : Flatten is used to convert the image data into 1D
'''
from keras.models import Sequential
from keras.layers import Dense, Flatten

flowerANN = Sequential()

flowerANN.add(Flatten())
```

```
                           #step_5.1 : adding layers
```

```
#256*256*3 - input dimensions(Xscaled shape)
flowerANN.add(Dense(units=1024, activation='relu'))
#hidden layer
flowerANN.add(Dense(units=350, activation='relu'))
#final layer - classification problem with 5 classes
flowerANN.add(Dense(units=5, activation='softmax'))
```

```
                            #step_5.2 : compile
```

```
flowerANN.compile(loss='categorical_crossentropy', metrics='accuracy', optimizer='adam')
```

step_6 : Saving best model

```
from keras.callbacks import ModelCheckpoint

mc = ModelCheckpoint(filepath='bestmodel.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
```

step_7 : fit

```
history = flowerANN.fit(Xtrain, ytrain, epochs=5, validation_data=(Xval, yval), callbacks=[mc])
```

```
Epoch 1/5
3/3 [==============================] - ETA: 0s - loss: 250.4520 - accuracy: 0.1351
Epoch 1: val_accuracy improved from -inf to 0.31579, saving model to bestmodel.h5
3/3 [==============================] - 14s 6s/step - loss: 250.4520 - accuracy: 0.1351 - val_loss: 297.1272 - val_accuracy: 0.3158
Epoch 2/5
2/3 [===================>..........] - ETA: 0s - loss: 321.4336 - accuracy: 0.2969
Epoch 2: val_accuracy did not improve from 0.31579
3/3 [==============================] - 0s 70ms/step - loss: 316.3593 - accuracy: 0.2838 - val_loss: 138.7765 - val_accuracy: 0.2105
Epoch 3/5
3/3 [==============================] - ETA: 0s - loss: 108.9163 - accuracy: 0.3784
Epoch 3: val_accuracy did not improve from 0.31579
3/3 [==============================] - 0s 69ms/step - loss: 108.9163 - accuracy: 0.3784 - val_loss: 186.1071 - val_accuracy: 0.2632
Epoch 4/5
```

```
2/3 [===================>..........] - ETA: 0s - loss: 169.4035 - accuracy: 0.2344
Epoch 4: val_accuracy did not improve from 0.31579
3/3 [==============================] - 0s 65ms/step - loss: 154.9585 - accuracy: 0.2568 - val_loss: 72.1773 - val_accuracy: 0.2105
Epoch 5/5
2/3 [===================>..........] - ETA: 0s - loss: 44.8381 - accuracy: 0.4375
Epoch 5: val_accuracy did not improve from 0.31579
3/3 [==============================] - 0s 61ms/step - loss: 45.4205 - accuracy: 0.4189 - val_loss: 85.9737 - val_accuracy: 0.3158
```

bayes error - trainingerror = avoidable bias

trainingerror - validation loss = variance

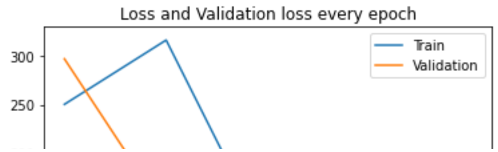
```
#ANN architecture

flowerANN.summary()
```

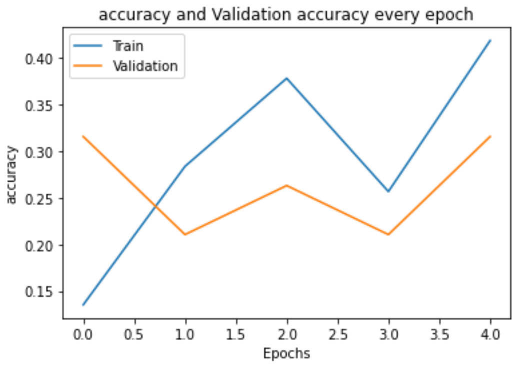
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 196608)            0         
                                                                 
 dense (Dense)               (None, 1024)              201327616 
                                                                 
 dense_1 (Dense)             (None, 350)               358750    
                                                                 
 dense_2 (Dense)             (None, 5)                 1755      
                                                                 
=================================================================
Total params: 201,688,121
Trainable params: 201,688,121
Non-trainable params: 0
_________________________________________________________________
```

```
import matplotlib.pyplot as plt
```

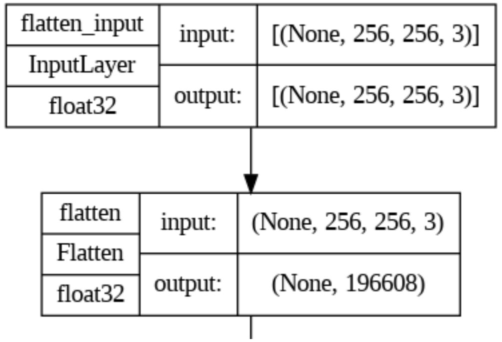
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.title('Loss and Validation loss every epoch')
plt.show()
```




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epochs')
plt.ylabel('accuracy')
plt.legend(['Train', 'Validation'])
plt.title('accuracy and Validation accuracy every epoch')
plt.show()
```




```
from tensorflow.keras.utils import plot_model
plot_model(flowerANN, show_shapes=True, show_dtype=True, show_layer_activations=True, show_layer_names=True)
```




```
ypred =flowerANN.predict(Xtest)
```

```
1/1 [==============================] - 0s 93ms/step
```

```
import numpy as np
ypredclasses = np.argmax(ypred, axis=-1)
```

```
ypredclasses
```

```
array([1, 1, 3, 3, 3, 1, 1, 3, 3, 3, 1, 1, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3,
       3, 1])
```

```
y_actual = enc.inverse_transform(ypredclasses)
```

```
y_actual
```

```
array(['dandelion', 'dandelion', 'sunflower', 'sunflower', 'sunflower',
       'dandelion', 'dandelion', 'sunflower', 'sunflower', 'sunflower',
       'dandelion', 'dandelion', 'sunflower', 'sunflower', 'dandelion',
       'sunflower', 'sunflower', 'sunflower', 'sunflower', 'sunflower',
       'sunflower', 'sunflower', 'sunflower', 'dandelion'], dtype='<U9')
```

```
from keras.models import load_model
bestmodel = load_model('/content/bestmodel.h5')
```

evaluate test set on final model

```
flowerANN.evaluate(Xtest, ytest)
```

```
1/1 [==============================] - 0s 33ms/step - loss: 62.6260 - accuracy: 0.3333
[62.62599182128906, 0.3333333432674408]
```

evaluate on bestmodel saved on checkpoint

```
bestmodel.evaluate(Xtest, ytest)

     1/1 [==============================] - 0s 121ms/step - loss: 248.1812 - accuracy: 0.2917
     [248.18115234375, 0.2916666567325592]


#ytrue_daisy =        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#y_pred =       [3, 3, 3, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1]
#y_self =       [0, 0, 0, 0, 0, 0, 1, 0, 3, 3, 0, 0, 0]

#y_pred_tulip = [ ]
'''
ytrue = [0,0,1,1]
yself = [1,0,1,0]
from sklearn.metrics import accuracy_score
accuracy_score(ytrue,yself)
#bayes acc - 68%

#ANN - metric=['accuracy']
#epoch

#training set acc -61%
bestmodel.predict(Xtrain)
#OR
bestmodel.evaluate(Xtrain, ytrain)

ypredtrain = [1,1,1,1]
accuracy_score(ypredtrain, ytrue)
#val set acc - 63%
bestmodel.predict(Xval)
#OR
bestmodel.evaluate(Xval, yval)
ypredval = [1,1,1,1]
accuracy_score(ypredval, ytrue)

#test set acc - 42%
ypredtest = [1,1,1,1]
accuracy_score(ypredtest, ytrue)

#Avoidable bias = 7%
#variance = Train - val = 2%
'''
'''
#am_df_cat = list(am_df.select_dtypes(include='object'))
#am_df_num = list(am_df.select_dtypes(exclude='object'))

## am_df_scaled.loc[am_df_scaled['horsepower']<0.33]['Bin'].value_counts()
## Boolean masking"

## am_df.groupby('body-style').agg(np.mean)['price']
'''

     '\n#am_df_cat = list(am_df.select_dtypes(include=\'object\'))\n#am_df_num = list(am_df.select_dtypes(e
     xclude=\'object\'))\n\n## am_df_scaled.loc[am_df_scaled[\'horsepower\']<0.33][\'Bin\'].value_counts()
     \n## Boolean masking"\n\n## am_df.groupby(\'body-style\').agg(np.mean)[\'price\']\n'
```

Predict for 1 image

```
#y_pred_tulip = [1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1]
#y_self_tulip = [2, 4, 4, 2, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4]
#y_true_tulip = [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
```

```
#read first file name
filepath = '/content/flowers/tulip/11746080_963537acdc.jpg'
#read image
img_read = cv2.imread(filepath, cv2.IMREAD_COLOR)
#resize image
img_resized = cv2.resize(img_read, (256, 256))
#covert to numpy
img_np = np.array(img_resized)
#add dimension
img_np_d = np.expand_dims(img_np, axis=0)
#shape
img_np_d.shape
#scale
img_np_d_scaled = img_np_d/255.0
```

```
bestmodel.predict(img_np_d_scaled)
```

```
1/1 [==============================] - 0s 18ms/step
array([[0.0000000e+00, 1.0000000e+00, 0.0000000e+00, 2.7259403e-11,
        0.0000000e+00]], dtype=float32)
```

```
import numpy as np
ypredclasses = np.argmax(bestmodel.predict(img_np_d_scaled), axis=-1)
```

```
1/1 [==============================] - 0s 18ms/step
```

```
ypredclasses
```

```
array([1])
```

```
y_actual = enc.inverse_transform(ypredclasses)
```

```
y_actual
```

```
array(['dandelion'], dtype='<U9')
```