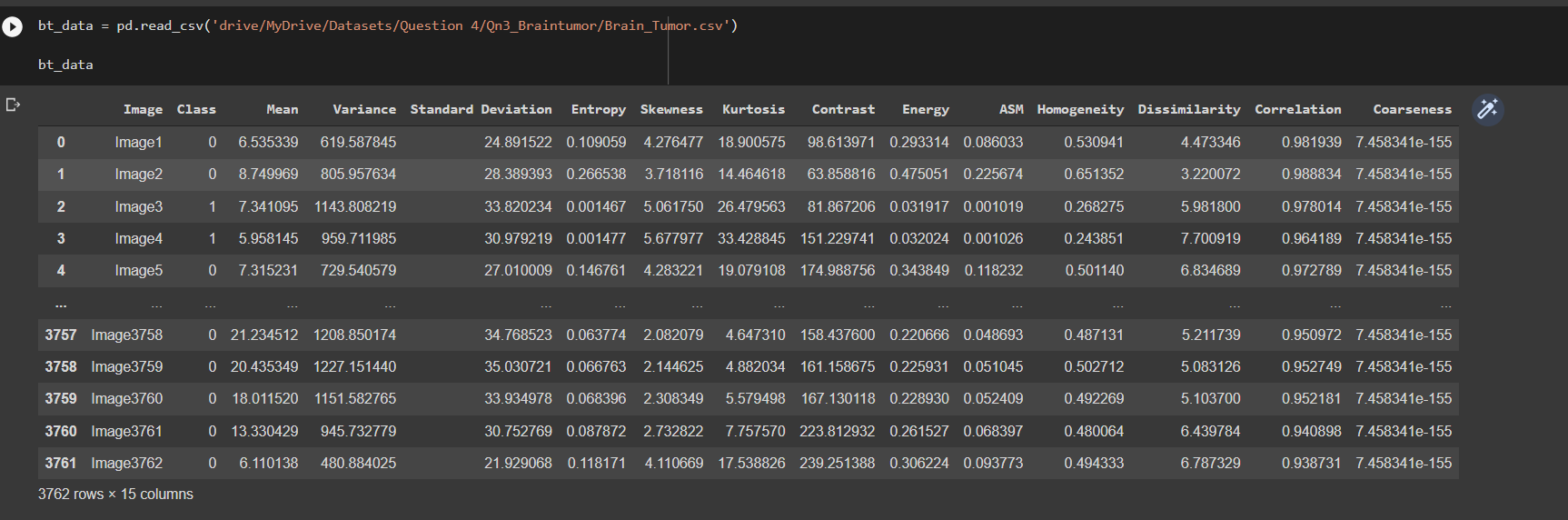
**Question.no : 4**

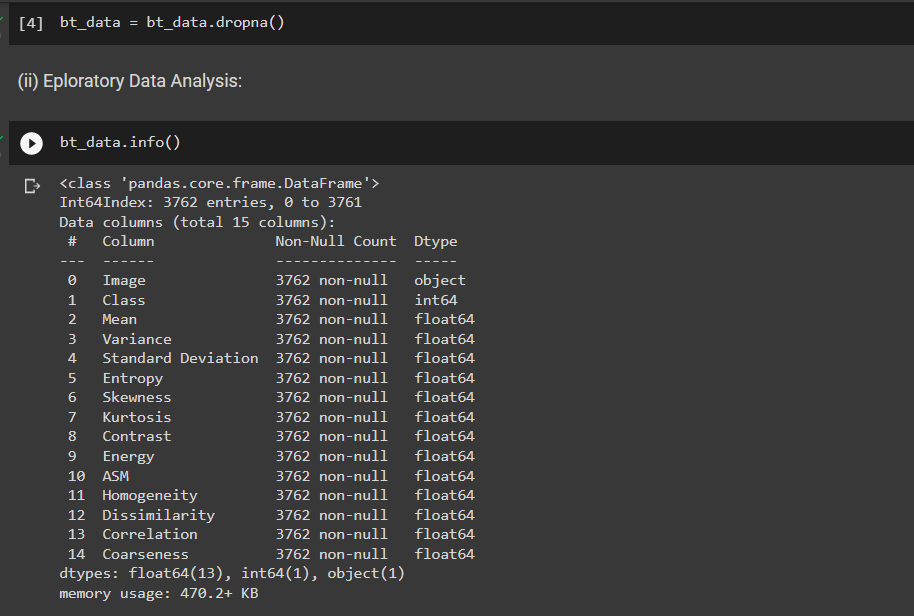
**BRAIN TUMOUR DETECTION**

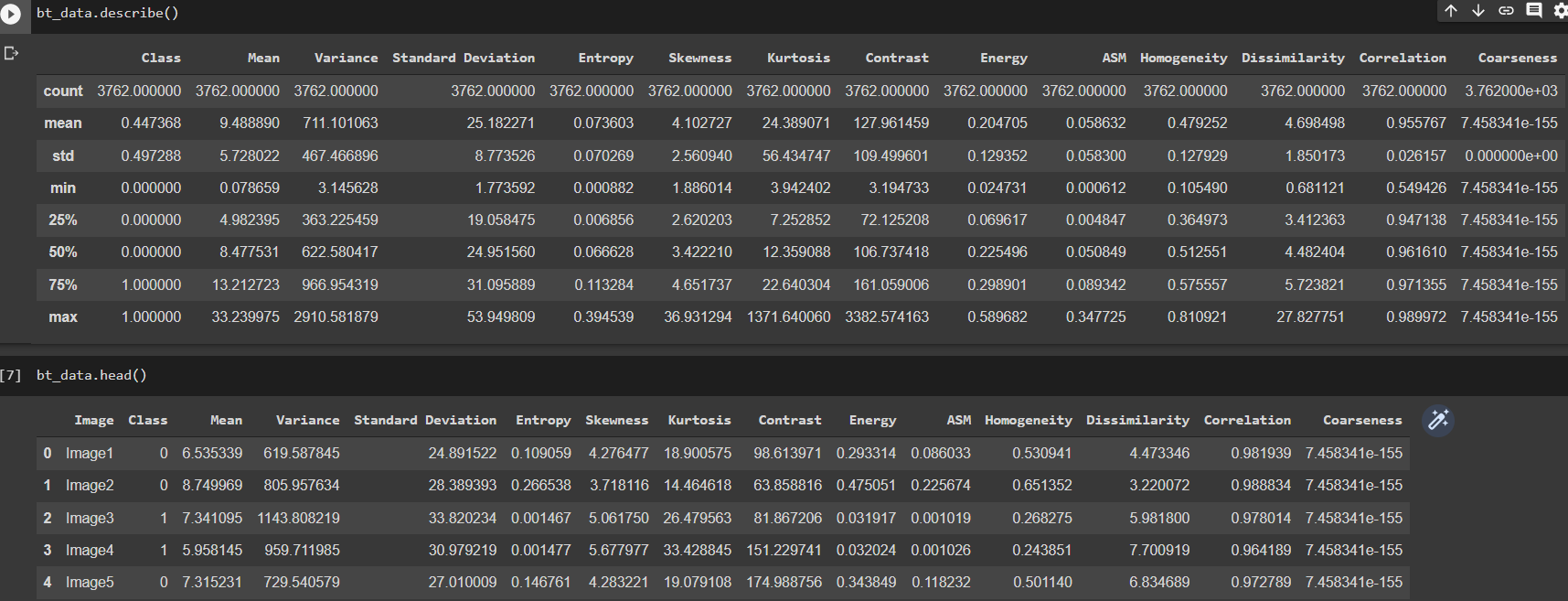
Procedure:

1. First imported the given csv file(Brain Tumour.csv)

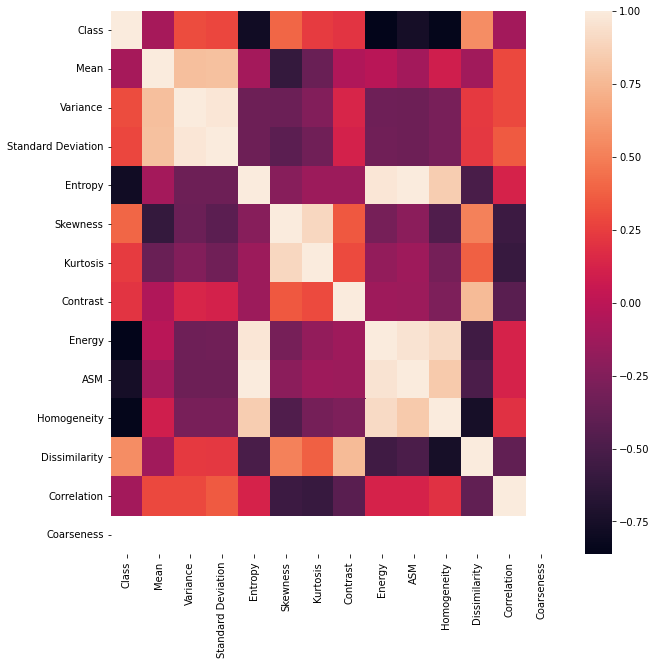


1. Then cleaned the dataset for Nan/Null values.
2. Exploratory data analysis is done based on the info of the dataset, description and correlation.

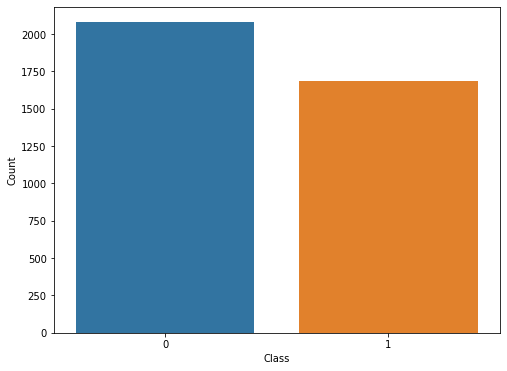




1. Correlation is visualized using heatmap to know which features are correlated the best.



1. Then the count of target class is visualized for checking the balance of dataset.[ If not balanced need to use SMOTE to equalize the target class count)



1. The X and Y values are assigned based on the prediction to be carried out.[ Y contains Class which is the target and X contains all other features .]
2. Then the X and Y is divided into X\_train, X\_test, Y\_train, Y\_test values for model fitting.
3. User-defined class for Logistic Regression is implemented and functions are included in it for training, prediction, activation function.

class LogitRegression() :

    def \_\_init\_\_( self, learning\_rate, iterations ) :

        self.learning\_rate = learning\_rate

        self.iterations = iterations

    # Function for model training

    def fit( self, X, Y ) :

        # no\_of\_training\_examples, no\_of\_features

        self.m, self.n = X.shape

        # weight initialization

        self.W = np.zeros( self.n )

        self.b = 0

        self.X = X

        self.Y = Y

        # gradient descent learning

        for i in range( self.iterations ) :

            self.update\_weights()

        return self

    # Helper function to update weights in gradient descent

    def update\_weights( self ) :

        A = 1 / ( 1 + np.exp( - ( self.X.dot( self.W ) + self.b ) ) )

        # calculate gradients

        tmp = ( A - self.Y.T )

        tmp = np.reshape( tmp, self.m )

        dW = np.dot( self.X.T, tmp ) / self.m

        db = np.sum( tmp ) / self.m

        # update weights

        self.W = self.W - self.learning\_rate \* dW

        self.b = self.b - self.learning\_rate \* db

        return self

    # Hypothetical function  h( x )

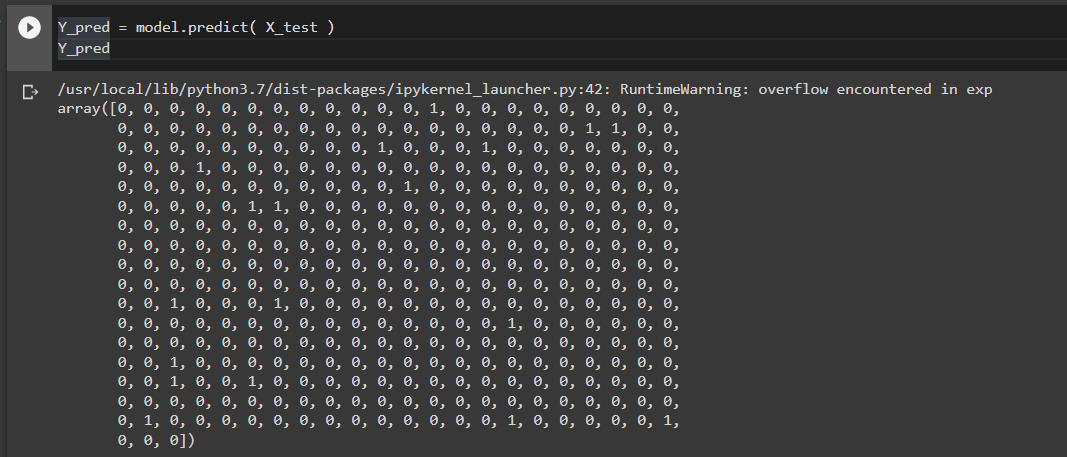
    def predict( self, X ) :

        Z = 1 / ( 1 + np.exp( - ( X.dot( self.W ) + self.b ) ) )

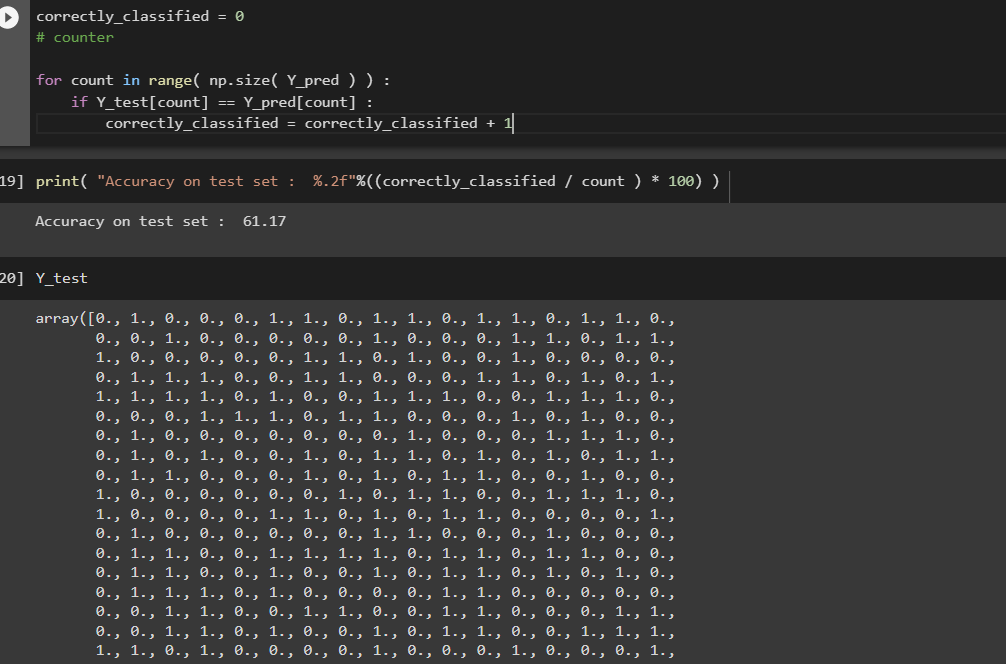
        Y = np.where( Z > 0.5, 1, 0 )

        return Y

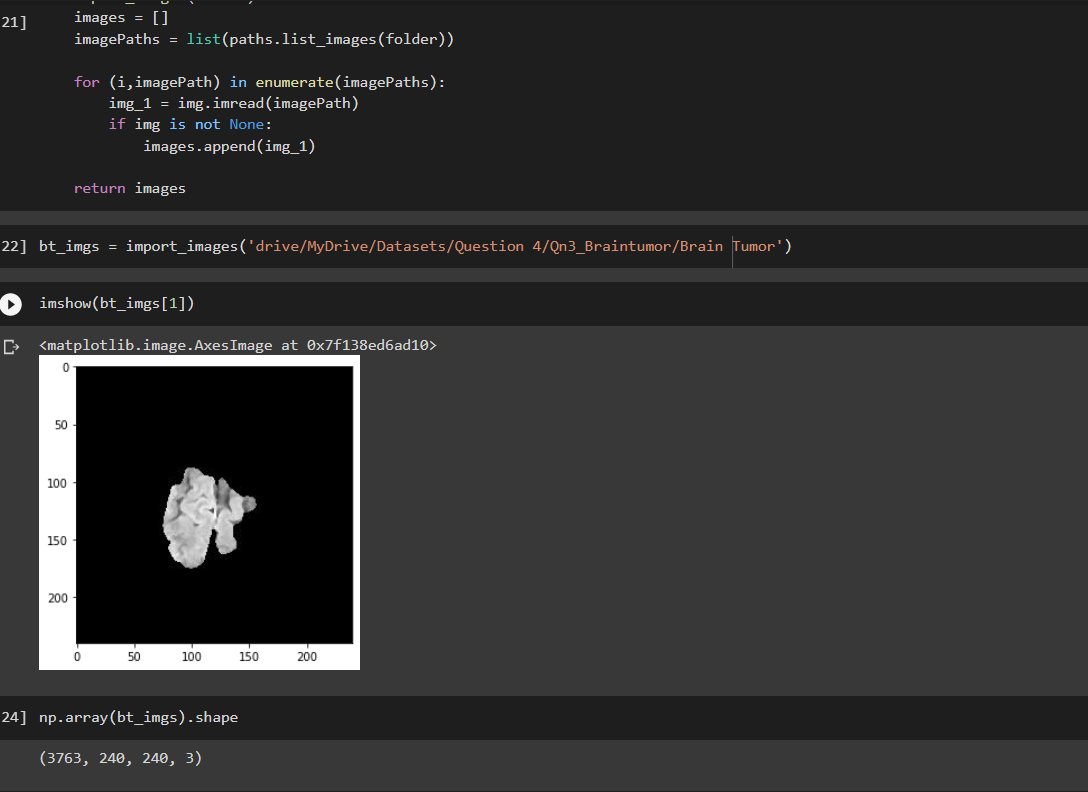
1. Then the model is trained and prediction is made.



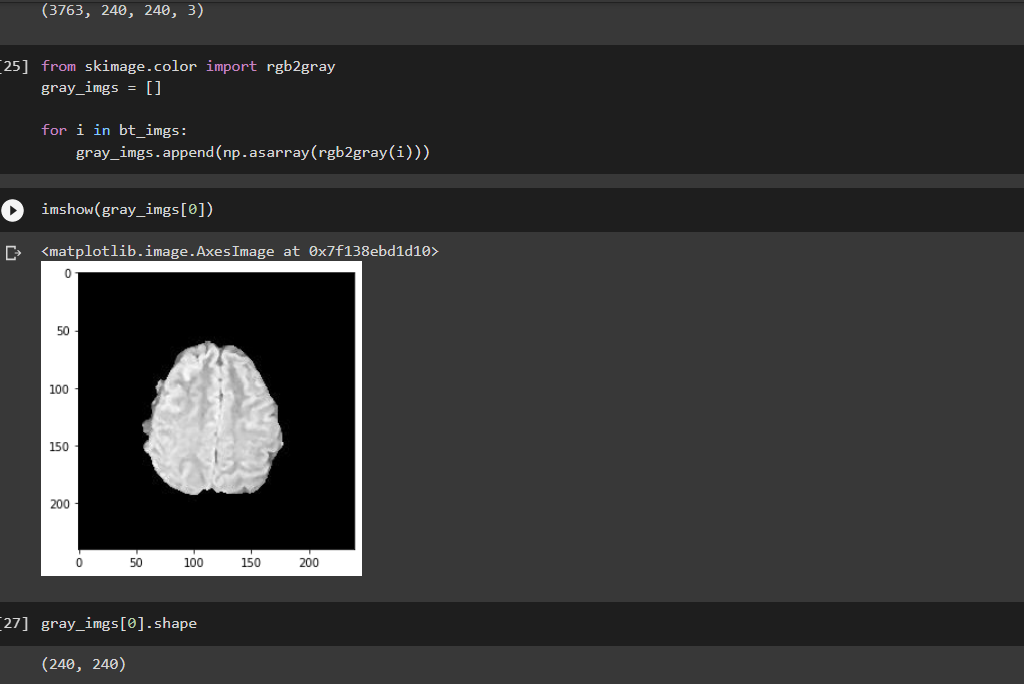
1. Accuracy is calculated based on correctly classified values on the test data.



1. For the second approach, images is imported and using that a neural network is implemented.



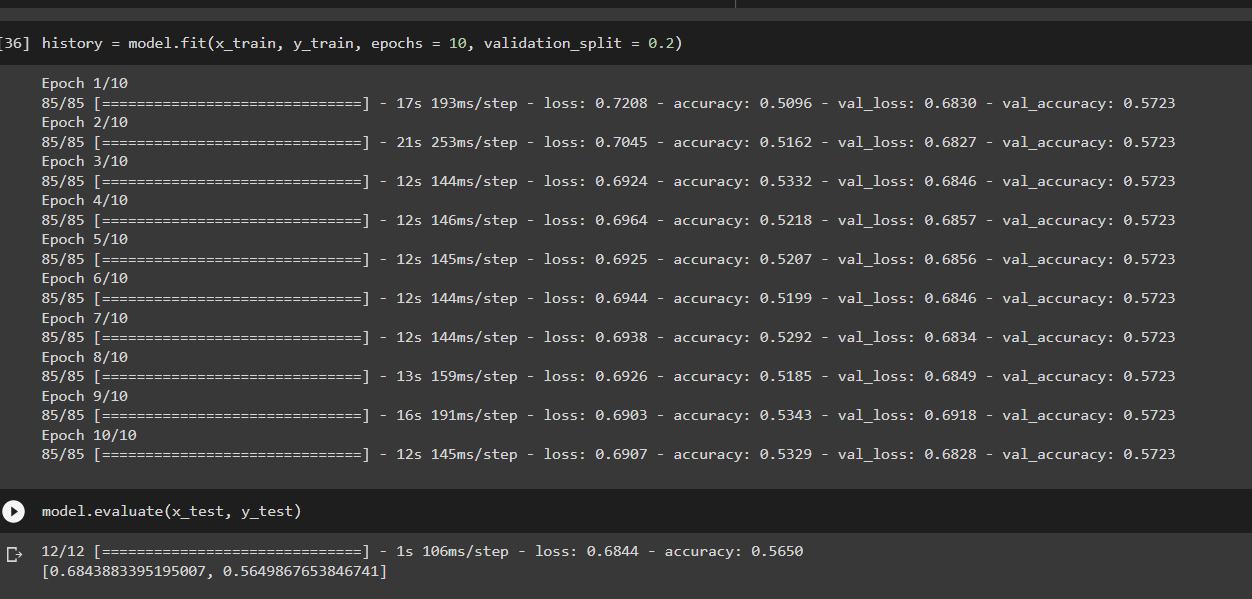
1. Here, the images is imported first and grayscaled to reduce the dimension of the image I.e., converting rgb-3d array to gray-2d array for better calculation.



1. Here also, the train and test data is divided.
2. A normal NN is implemented with input size as 240x240 and with 3 hidden layers.
3. Normally, Relu is used for activation function since it is reliable and feasible.
4. Tanh activation function is used for better perception of the NN.



1. The model is trained using the above set values using the training data.
2. Then the model is tested against the testing values and accuracy is found.





1. Using the validation set, the train and validate plot is derived.

