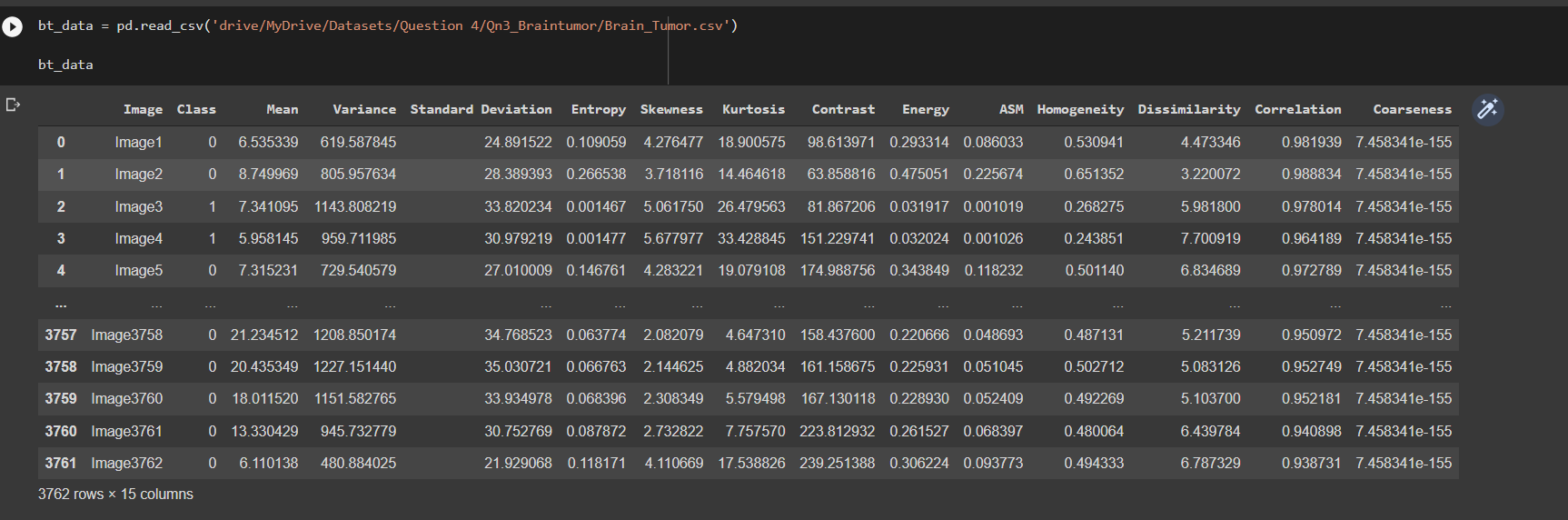
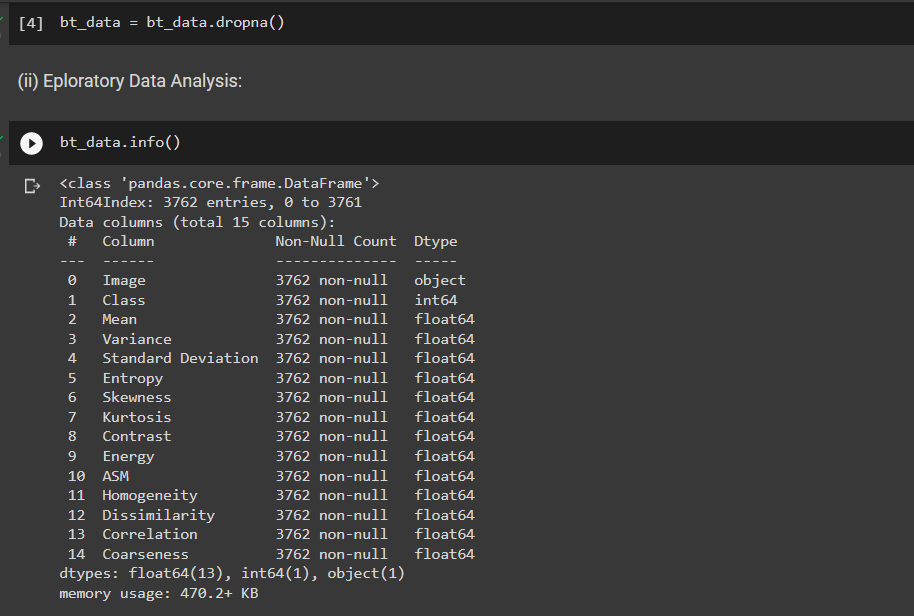
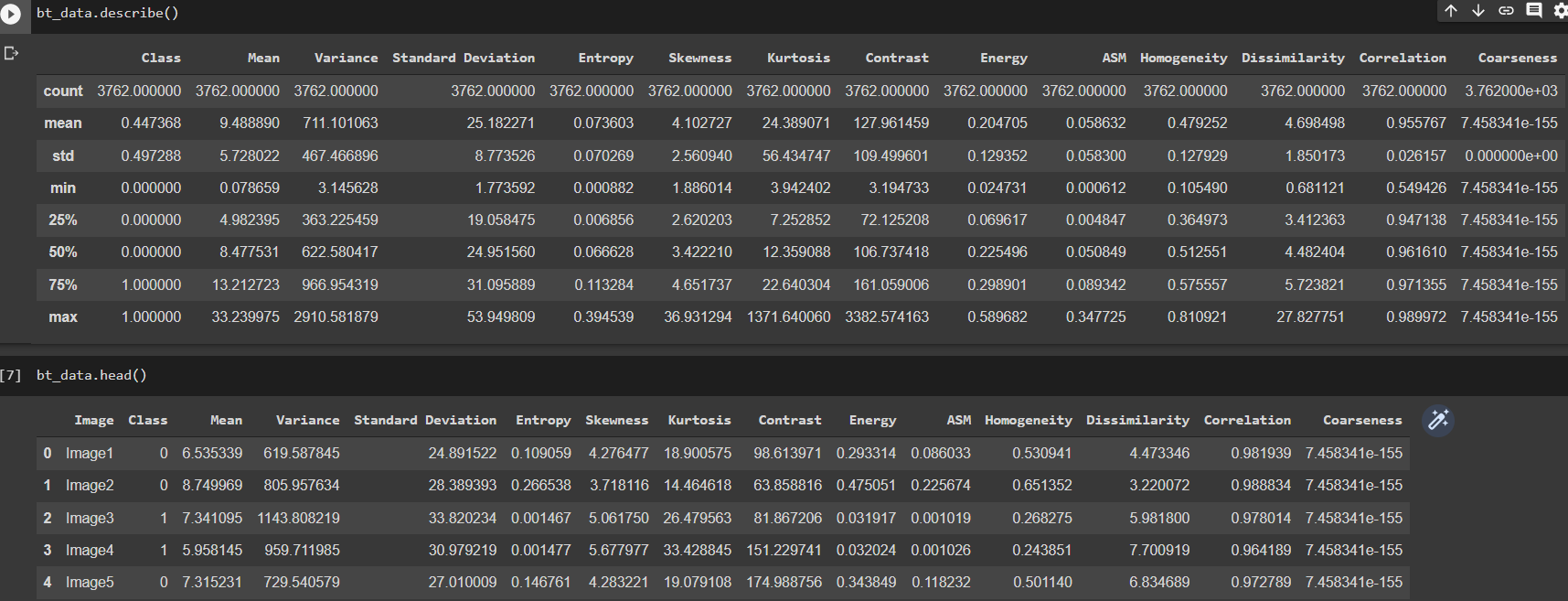
**Question.no : 4**

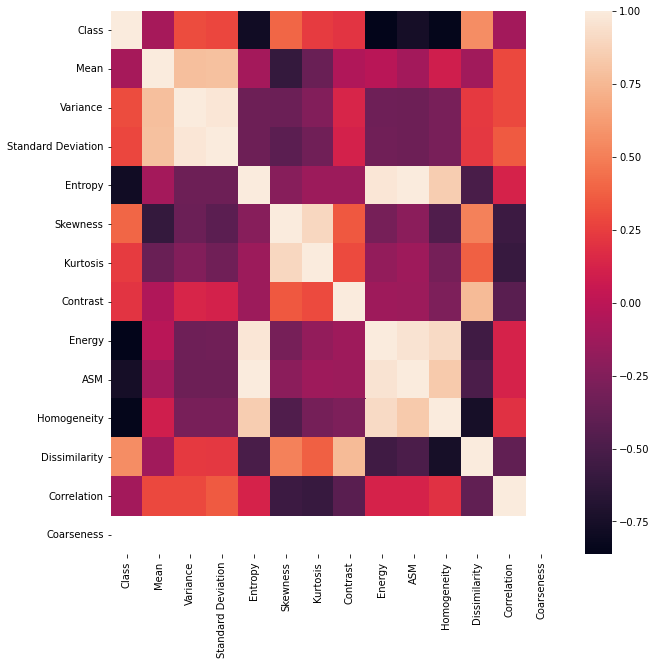
**BRAIN TUMOUR DETECTION**

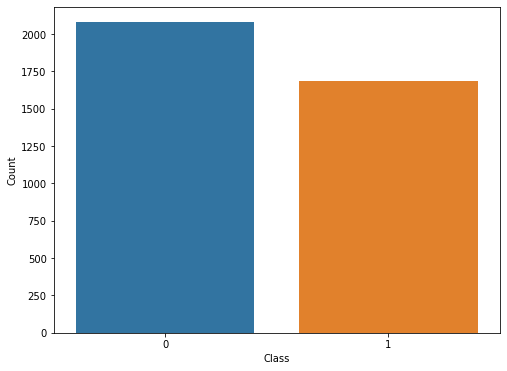
Screenshot of Outputs:











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

