In [161…

```
import matplotlib.pyplot as plt                          #importing various libraries
import matplotlib.image as mpimg
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import numpy as np
import seaborn as sns
import pandas as pd
import glob
import cv2
import os
from imutils import paths
from scipy import ndimage as nd
from skimage.feature import graycomatrix, graycoprops
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to
import torchio as tio
from sklearn.cluster import KMeans
from sklearn.model_selection import GridSearchCV
```

Data Augmentation using Keras:

In [298…

```
dataGen = ImageDataGenerator(
        zoom_range= 0.5, vertical_flip = True
)


imagesBT = []


path = "/Users/alimi/Downloads/archive (2)/Brain Tumor Data Set/Brain Tumor Data Set/
list(os.listdir(path))
imagePaths = list(paths.list_images(path))

for imagePath in imagePaths:
        imageBT = cv2.imread(imagePath)
        imageBT = cv2.cvtColor(imageBT, cv2.COLOR_BGR2GRAY)
        imageBT = cv2.resize(imageBT, (255,255))
        imagesBT.append(imageBT)

images_BTarray = np.array(imagesBT)
images_BTarray = images_BTarray.reshape(images_BTarray.shape + (1,))
i = 0
for batch in dataGen.flow(images_BTarray, batch_size=10,
                          save_to_dir='/Users/alimi/Downloads/archive (2)/Brain Tumo
                          save_prefix='aug2',
                          save_format='jpg'):
    i += 1
    if i > 100:
         break


imagesNC = []
path = "/Users/alimi/Downloads/archive (2)/Brain Tumor Data Set/Brain Tumor Data Set/
list(os.listdir(path))
imagePaths = list(paths.list_images(path))

for imagePath in imagePaths:
    imageNC = cv2.imread(imagePath)
    imageNC = cv2.cvtColor(imageNC, cv2.COLOR_BGR2GRAY)
```

```
    imageNC = cv2.resize(imageNC, (255,255))
    imagesNC.append(imageNC)

images_NCarray = np.array(imagesNC)
images_NCarray = images_NCarray.reshape(images_NCarray.shape + (1,))
i = 0
for batch in dataGen.flow(images_NCarray, batch_size=10,
                          save_to_dir='/Users/alimi/Downloads/archive (2)/Brain Tumor
                          save_prefix='aug2',
                          save_format='jpg'):
    i += 1
    if i > 100:
        break
```

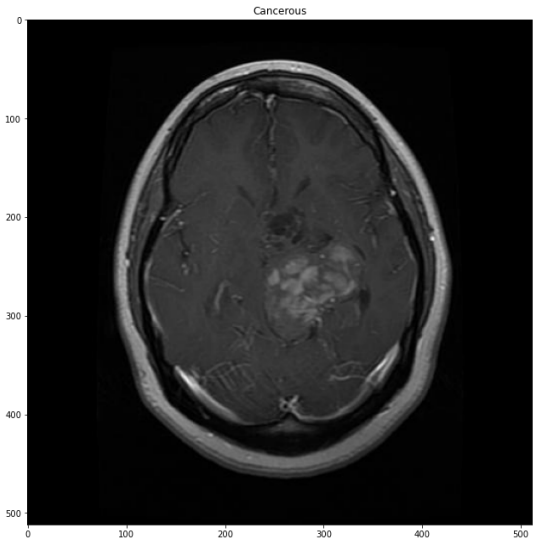
In [151…

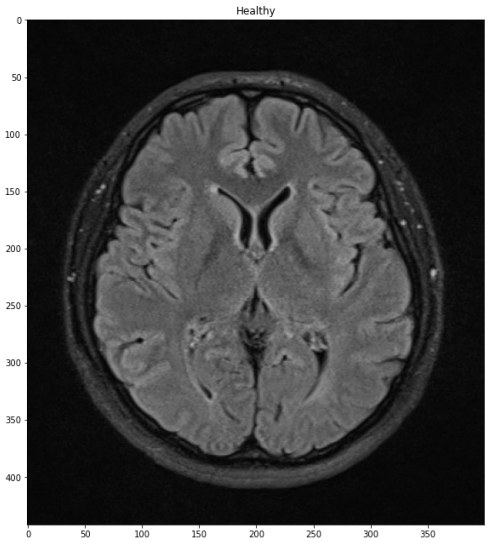
```
images = []
labels = []
path = "/Users/alimi/Downloads/archive (2)/Brain Tumor Data Set/Brain Tumor Data Set/
list(os.listdir(path))                    #Here, each image's pixel data and correspo
imagePaths = list(paths.list_images(path)) #are appended to 2 numpy arrays

for imagePath in imagePaths:
    label = imagePath.split(os.path.sep)[-2]
    image = cv2.imread(imagePath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (255,255))
    images.append(image)
    labels.append(label)

images_array = np.array(images)
labels_array = np.array(labels)
print(images_array.shape)
```

```
(30029, 255, 255)
```

In [152…

```
plt.figure(figsize=(32, 24))     #Viewing examples of MRI images showing presence/non

fileNames = ['Brain Tumor/Cancer (1).jpg', 'Healthy/7.jpg']
fileLabels = ['Cancerous', 'Healthy']

for i in range(2):
    ax = plt.subplot(2, 2, i + 1)
    img = mpimg.imread(path + fileNames[i])
    plt.imshow(img)
    plt.title(fileLabels[i])
```



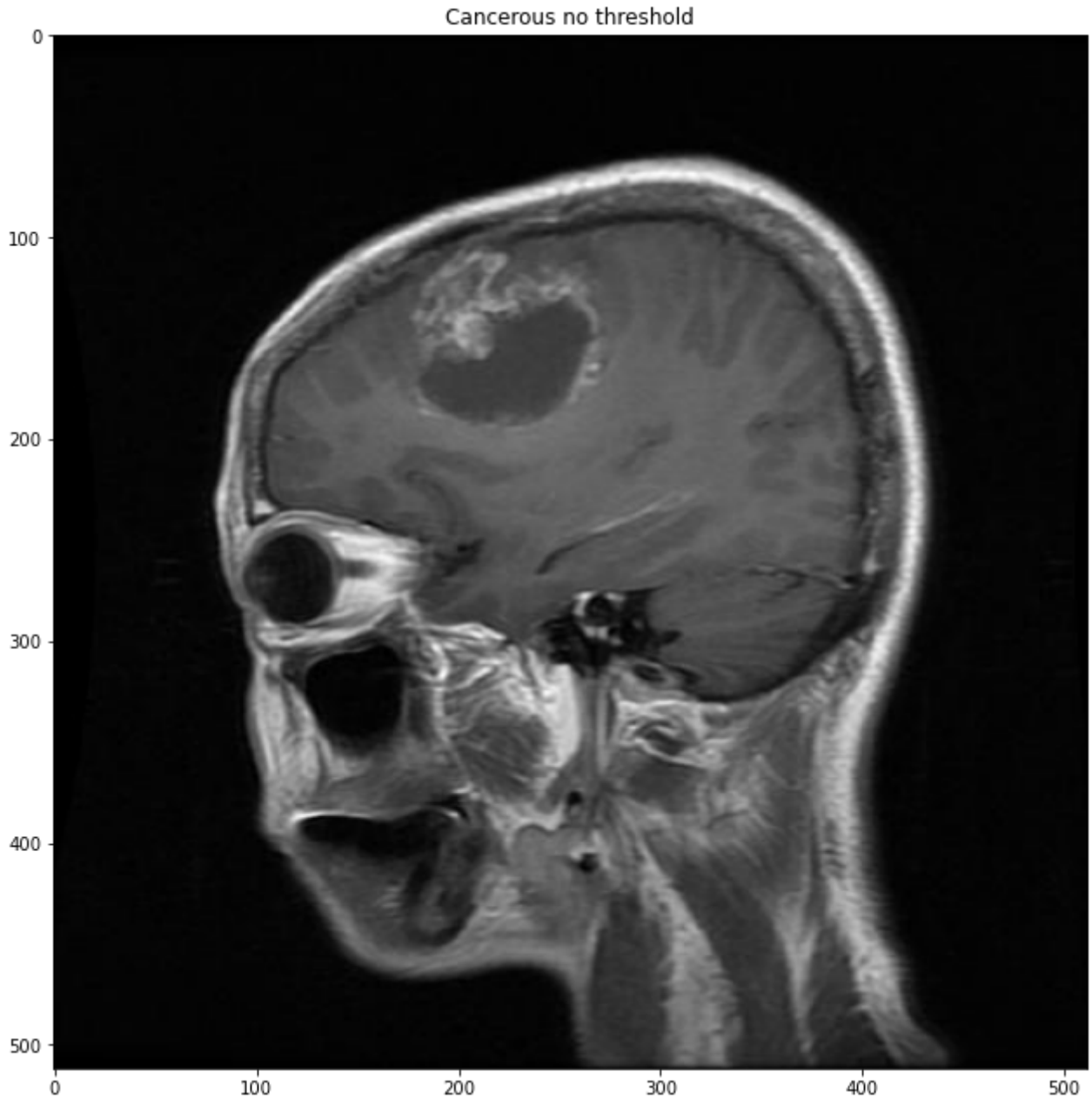


In [153…

```
plt.figure(figsize=(32, 24))
```

```
fileNames = ['Brain Tumor/Cancer (218).jpg', 'Brain Tumor/Cancer (216).jpg']
fileLabels = ['Cancerous no threshold', 'With threshold']
ax = plt.subplot(2, 2, 1)
img = mpimg.imread(path + fileNames[0])
plt.imshow(img)
plt.title(fileLabels[0])
```

Out[153]: Text(0.5, 1.0, 'Cancerous no threshold')



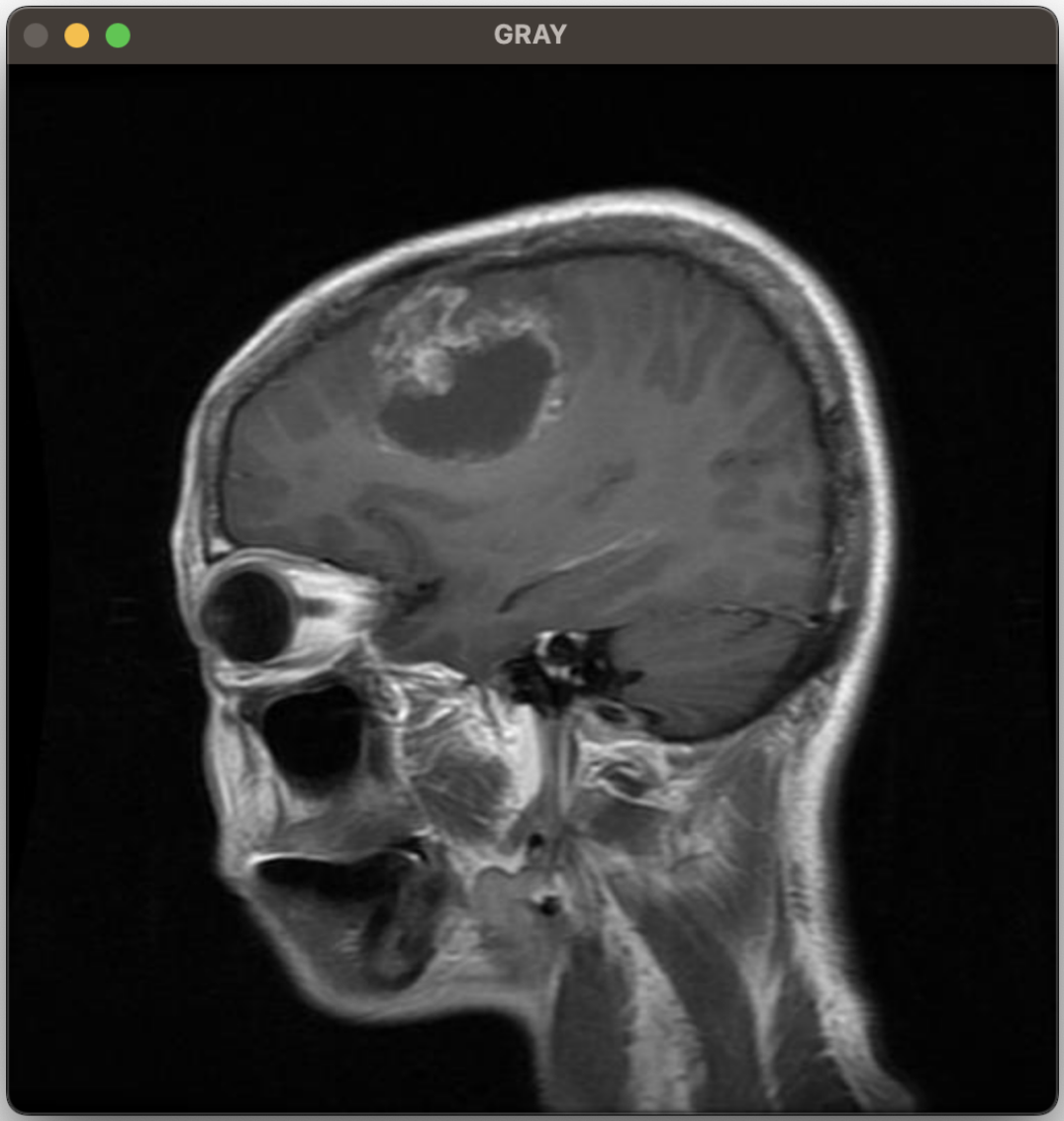
Viewing Examples from Applying Thresholding:

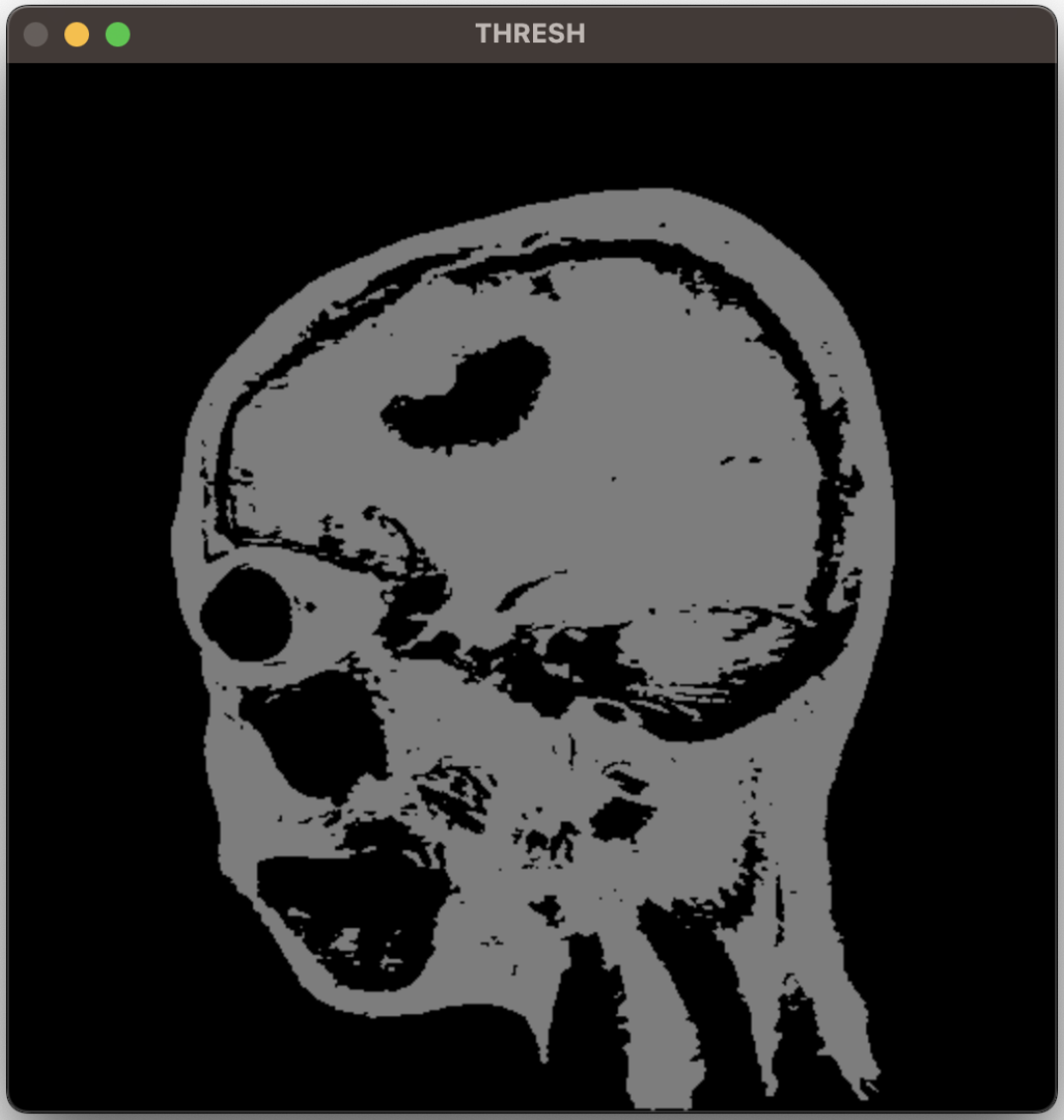
In [109…

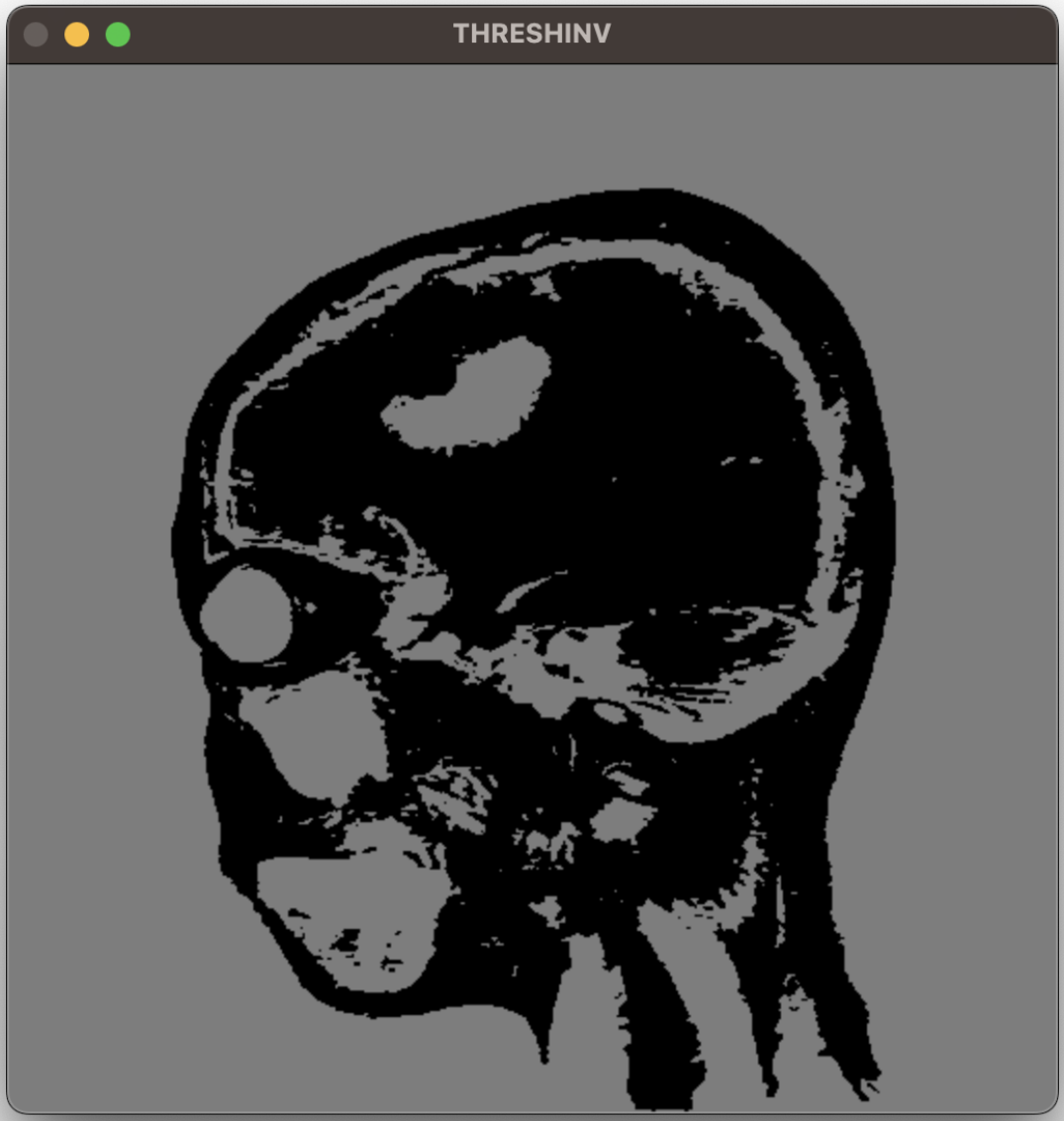
```
imgThresh = cv2.imread("/Users/alimi/Downloads/archive (2)/Brain Tumor Data Set/Brain
gray = cv2.cvtColor(imgThresh, cv2.COLOR_BGR2GRAY)
cv2.imshow("GRAY", gray)
cv2.waitKey(0);
cv2.destroyAllWindows();
cv2.waitKey(1)
(T, thresh) = cv2.threshold(gray, 70, 125, cv2.THRESH_BINARY)
cv2.imshow("THRESH",thresh)
cv2.waitKey(0);
cv2.destroyAllWindows();
cv2.waitKey(1)
(T, threshInv) = cv2.threshold(gray, 70, 125,cv2.THRESH_BINARY_INV)
cv2.imshow("THRESHINV",threshInv)
```

```
cv2.waitKey(0);
cv2.destroyAllWindows();
cv2.waitKey(1)
```

Out[109]: -1




THRESH



In [154…
```
from sklearn import preprocessing   #Here, class labels are assigned "0" = Brain Tumo
le = preprocessing.LabelEncoder()
le.fit(labels_array)
LabelsEncoded = le.transform(labels_array)
print(LabelsEncoded.size)
```

```
30029
```

In [155…
```
#Splitting the dataset into a train(75%) and test(25%) set
x_train, x_test, y_train, y_test = train_test_split(images_array,LabelsEncoded,train_
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(22521, 255, 255)
(7508, 255, 255)
(22521,)
(7508,)
```

GLCM Feature Extraction

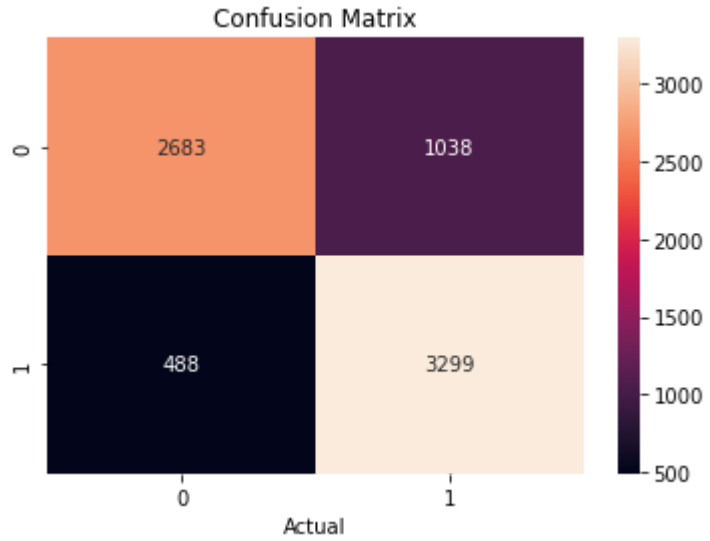
In [156…
```
def featureExtraction(dataset):
    imageDataset = pd.DataFrame()
    for image in range(dataset.shape[0]):
        df = pd.DataFrame()
        img = dataset[image, :, :]
        for i,j in [(1,0), (3,0), (5,0), (0,np.pi/4), (0,np.pi/2)]:
            GLCM = graycomatrix(img, [i], [j])
            Energy = graycoprops(GLCM, 'energy')[0]
            df['Energy'] = Energy
            Correlation = graycoprops(GLCM, 'correlation')[0]
            df['Correlation'] = Correlation
            Dissimilarity = graycoprops(GLCM, 'dissimilarity')[0]
            df['Dissimilarity'] = Dissimilarity
            Homogeneity = graycoprops(GLCM, 'homogeneity')[0]
            df['Homogeneity'] = Homogeneity
            Contrast = graycoprops(GLCM, 'contrast')[0]
            df['Contrast'] = Contrast
            imageDataset = imageDataset.append(df)




    return imageDataset
```

In [157…
```
image_features = featureExtraction(x_train)
XTrain = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTrain = np.reshape(image_features, (x_train.shape[0], -1))
```

In [158…
```
from sklearn import svm
model = svm.SVC(gamma = 'auto')
model.fit(XTrain, y_train)
```

Out[158]: SVC(gamma='auto')

In [164…
```
image_features = featureExtraction(x_test)
XTest = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTest = np.reshape(image_features, (x_test.shape[0], -1))
pred = model.predict(XTest)
cmat = confusion_matrix(y_test, pred, labels = [0,1])
s = sns.heatmap(cmat,annot = True, fmt = 'g',xticklabels= [0,1],yticklabels= [0,1])
plt.xlabel("Predicted")
plt.xlabel("Actual")
plt.title("Confusion Matrix")
plt.show()
acc = cmat.trace()/cmat.sum()
print('Accuracy: {0:5.2f}%'.format(acc*100))
```




```
Accuracy: 79.68%
```

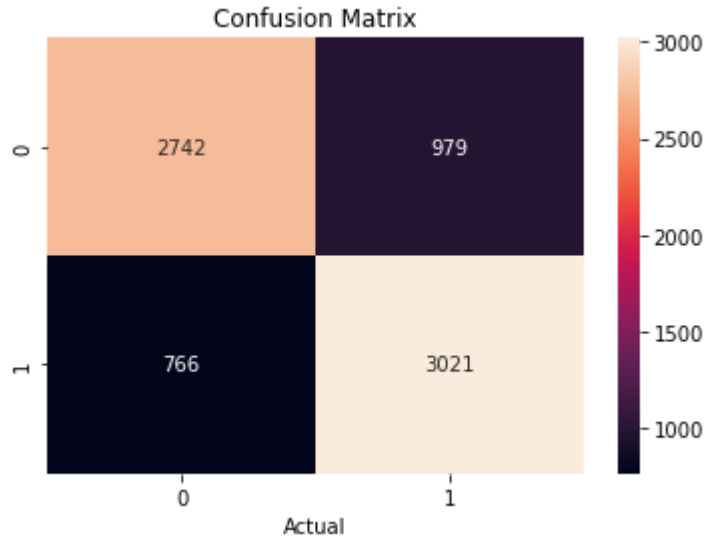
In [53]:
```
parameters = {'C': [10,100], 'kernel': ['linear']}
grid = GridSearchCV(svm.SVC(), parameters)
grid.fit(XTrain, y_train)

print(grid.best_params_)
print(grid.score(XTest, y_test))
```

```
{'C': 10, 'kernel': 'linear'}
0.7675812466702184
```

In [165…
```
from sklearn import svm
model = svm.SVC(C = 10, kernel = 'linear')
model.fit(XTrain, y_train)
```

Out[165]:
```
SVC(C=10, kernel='linear')
```

In [166…
```
image_features = featureExtraction(x_test)
XTest = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTest = np.reshape(image_features, (x_test.shape[0], -1))
pred = model.predict(XTest)
cmat = confusion_matrix(y_test, pred, labels = [0,1])
s = sns.heatmap(cmat,annot = True, fmt = 'g',xticklabels= [0,1],yticklabels= [0,1])
plt.xlabel("Predicted")
plt.xlabel("Actual")
plt.title("Confusion Matrix")
plt.show()
acc = cmat.trace()/cmat.sum()
print('Accuracy: {0:5.2f}%'.format(acc*100))
```




```
Accuracy: 76.76%
```

In [50]:
```
kernel = ['linear', 'rbf', 'poly', 'sigmoid']
for i in kernel:
    model = svm.SVC(kernel = i, C = 1.0)
    model.fit(XTrain, y_train)
    print("For kernel", i)
    image_features = featureExtraction(x_test)
    XTest = image_features
    n_features = image_features.shape[1]
    image_features = np.expand_dims(image_features, axis=0)
    XTest = np.reshape(image_features, (x_test.shape[0], -1))
    pred = model.predict(XTest)
    cmat = metrics.confusion_matrix(pred, y_test)
    acc = cmat.trace()/cmat.sum()
    print('Accuracy: {0:5.2f}%'.format(acc*100))
```

```
For kernel linear
Accuracy: 76.70%
For kernel rbf
Accuracy: 67.29%
For kernel poly
Accuracy: 53.17%
For kernel sigmoid
Accuracy: 43.18%
```

In [168…
```
from sklearn.ensemble import RandomForestClassifier
RF_model = RandomForestClassifier(n_estimators = 60, random_state = 42)
RF_model.fit(XTrain, y_train)
```

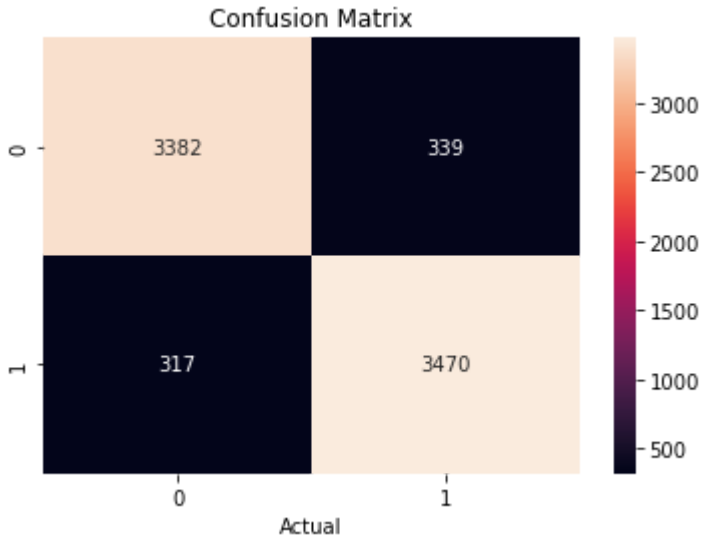
Out[168]:
```
RandomForestClassifier(n_estimators=60, random_state=42)
```

In [169…
```
image_features = featureExtraction(x_test)
XTest = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTest = np.reshape(image_features, (x_test.shape[0], -1))
```

In [170…
```
pred = RF_model.predict(XTest)
```

In [171…
```
cmat = confusion_matrix(y_test, pred, labels = [0,1])
s = sns.heatmap(cmat,annot = True, fmt = 'g',xticklabels= [0,1],yticklabels= [0,1])
plt.xlabel("Predicted")
plt.xlabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
acc = cmat.trace()/cmat.sum()
print('Accuracy: {0:5.2f}%'.format(acc*100))
```




```
Accuracy: 91.26%
```

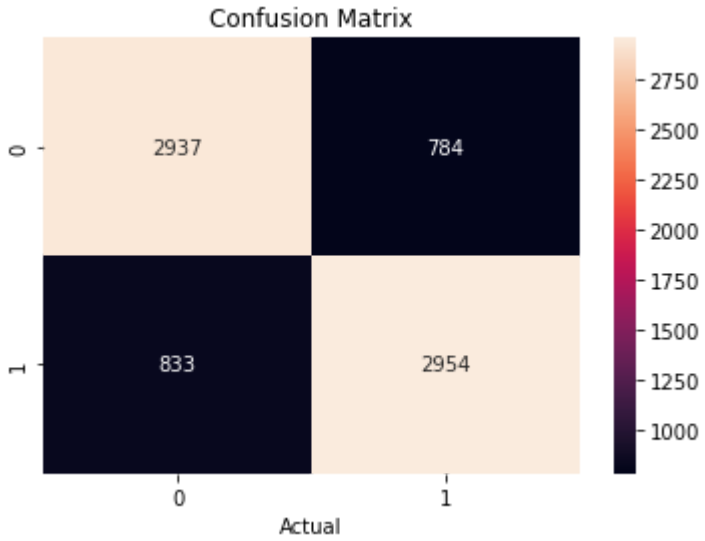
In [177…
```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier()
```

In [178…
```
KNN.fit(XTrain, y_train)
```

Out[178]:
```
KNeighborsClassifier()
```

In [175…
```
image_features = featureExtraction(x_test)
XTest = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTest = np.reshape(image_features, (x_test.shape[0], -1))
pred = KNN.predict(XTest)
```

In [176…
```
cmat = confusion_matrix(y_test, pred, labels = [0,1])
s = sns.heatmap(cmat,annot = True, fmt = 'g',xticklabels= [0,1],yticklabels= [0,1])
plt.xlabel("Predicted")
plt.xlabel("Actual")
plt.title("Confusion Matrix")
plt.show()
acc = cmat.trace()/cmat.sum()
print('Accuracy: {0:5.2f}%'.format(acc*100))
```




```
Accuracy: 78.46%
```

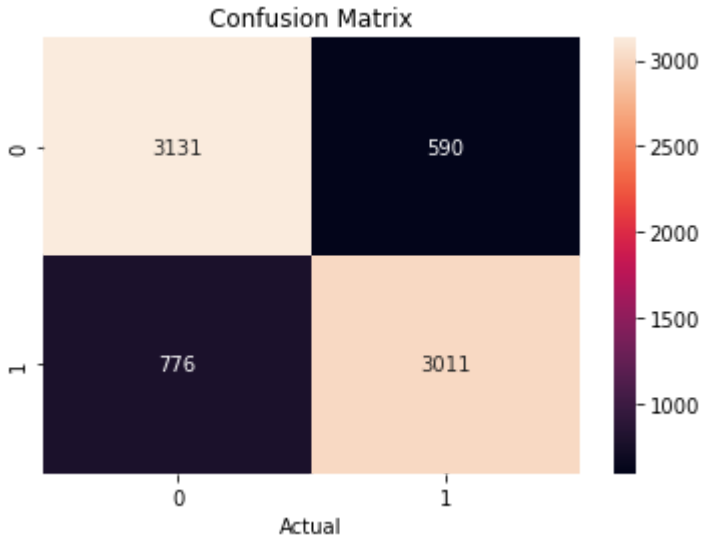
In [179…
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto'],
              'p': [1,2]}
knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(XTrain,y_train)
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 1,
'p': 1}
accuracy : 0.8164824244603013
```

In [180…
```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(algorithm = 'auto', n_neighbors = 1, p = 1)
KNN.fit(XTrain, y_train)
```

Out[180]:
```
KNeighborsClassifier(n_neighbors=1, p=1)
```

In [181…
```
image_features = featureExtraction(x_test)
XTest = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTest = np.reshape(image_features, (x_test.shape[0], -1))
pred = KNN.predict(XTest)
cmat = confusion_matrix(y_test, pred, labels = [0,1])
s = sns.heatmap(cmat,annot = True, fmt = 'g',xticklabels= [0,1],yticklabels= [0,1])
plt.xlabel("Predicted")
plt.xlabel("Actual")
plt.title("Confusion Matrix")
plt.show()
acc = cmat.trace()/cmat.sum()
print('Accuracy: {0:5.2f}%'.format(acc*100))
```




```
Accuracy: 81.81%
```

With Image Processing: Gaussian Filter, Thresholding and Morphological Operations (Erosion/Dilation) for each classifier.

In [182…
```
images = []
labels = []
path = "/Users/alimi/Downloads/archive (2)/Brain Tumor Data Set/Brain Tumor Data Set/
list(os.listdir(path))                    #Here, each image's pixel data and correspo
```

```
imagePaths = list(paths.list_images(path)) #are appended to 2 numpy arrays

for imagePath in imagePaths:
    label = imagePath.split(os.path.sep)[-2]
    image = cv2.imread(imagePath)
    #Applying gaussian filter
    image = nd.gaussian_filter(image, sigma = 5)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #Thresholding
    (T, image) = cv2.threshold(image, 70, 125,cv2.THRESH_BINARY_INV)
    #Morphological Operations
    kernel = np.ones((3,3), np.uint8)
    image = cv2.erode(image, kernel, iterations = 1)
    image = cv2.dilate(image, kernel, iterations = 1)
    image = cv2.resize(image, (255,255))
    images.append(image)
    labels.append(label)

images_array = np.array(images)
labels_array = np.array(labels)
print(images_array.shape)
```

```
(30029, 255, 255)
```

In [183…
```
from sklearn import preprocessing   #Here, class labels are assigned "0" = Brain Tumo
le = preprocessing.LabelEncoder()
le.fit(labels_array)
LabelsEncoded = le.transform(labels_array)
print(LabelsEncoded.size)
```

```
30029
```

In [184…
```
#Splitting the dataset into a train(75%) and test(25%) set
x_train, x_test, y_train, y_test = train_test_split(images_array,LabelsEncoded,train_
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(22521, 255, 255)
(7508, 255, 255)
(22521,)
(7508,)
```

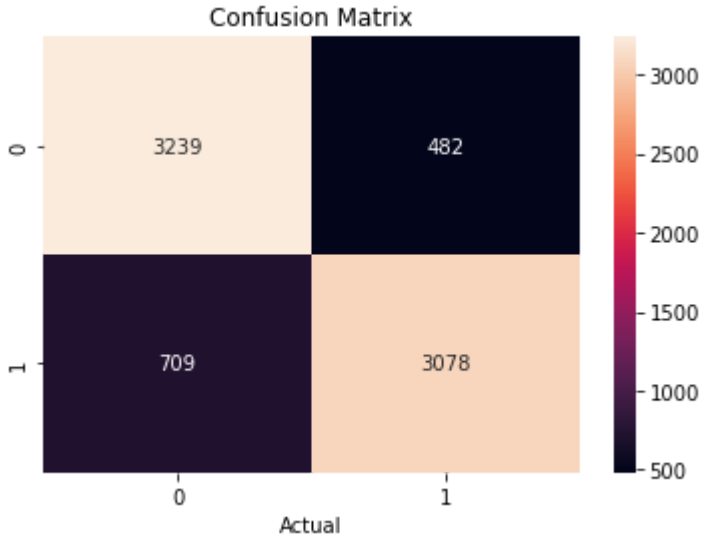
In [185…
```
image_features = featureExtraction(x_train)
XTrain = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTrain = np.reshape(image_features, (x_train.shape[0], -1))
```

In [186…
```
from sklearn import svm
model = svm.SVC(gamma = 'auto')
model.fit(XTrain, y_train)
```

Out[186]:
```
SVC(gamma='auto')
```

In [187…
```
image_features = featureExtraction(x_test)
XTest = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTest = np.reshape(image_features, (x_test.shape[0], -1))
pred = model.predict(XTest)
cmat = confusion_matrix(y_test, pred, labels = [0,1])
s = sns.heatmap(cmat,annot = True, fmt = 'g',xticklabels= [0,1],yticklabels= [0,1])
```

```
plt.xlabel("Predicted")
plt.xlabel("Actual")
plt.title("Confusion Matrix")
plt.show()
acc = cmat.trace()/cmat.sum()
print('Accuracy: {0:5.2f}%'.format(acc*100))
```




```
Accuracy: 84.14%
```

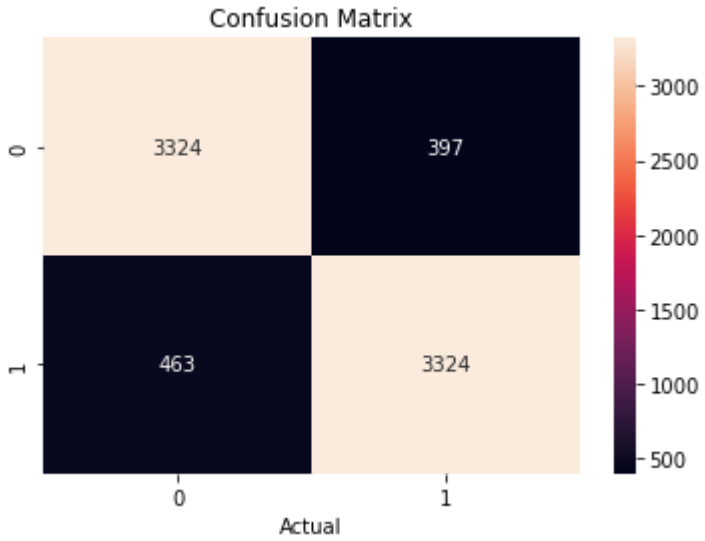
In [196…
```
from sklearn.ensemble import RandomForestClassifier
RF_model = RandomForestClassifier(n_estimators = 60, random_state = 42)
RF_model.fit(XTrain, y_train)
```

Out[196]:
```
RandomForestClassifier(n_estimators=60, random_state=42)
```

In [197…
```
image_features = featureExtraction(x_test)
XTest = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTest = np.reshape(image_features, (x_test.shape[0], -1))
```

In [198…
```
pred = RF_model.predict(XTest)
```

In [199…
```
cmat = confusion_matrix(y_test, pred, labels = [0,1])
s = sns.heatmap(cmat,annot = True, fmt = 'g',xticklabels= [0,1],yticklabels= [0,1])
plt.xlabel("Predicted")
plt.xlabel("Actual")
plt.title("Confusion Matrix")
plt.show()
acc = cmat.trace()/cmat.sum()
print('Accuracy: {0:5.2f}%'.format(acc*100))
```



Accuracy: 88.55%

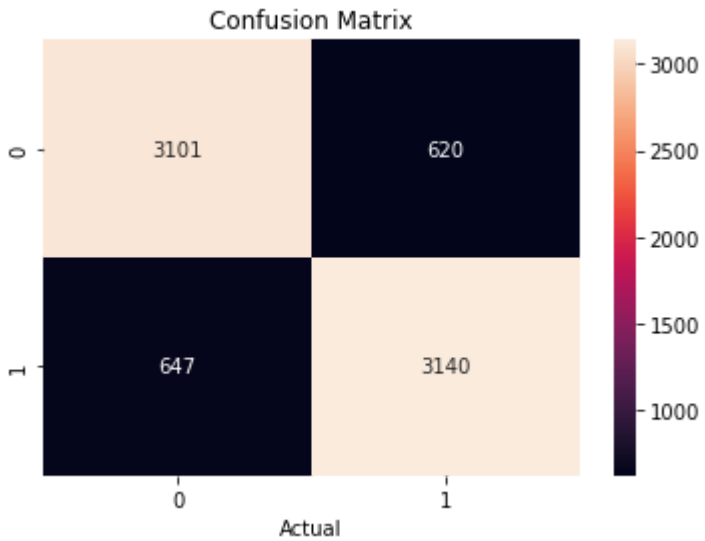
In [192…
```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier()
```

In [193…
```
KNN.fit(XTrain, y_train)
```

Out[193]: KNeighborsClassifier()

In [194…
```
image_features = featureExtraction(x_test)
XTest = image_features
n_features = image_features.shape[1]
image_features = np.expand_dims(image_features, axis=0)
XTest = np.reshape(image_features, (x_test.shape[0], -1))
pred = KNN.predict(XTest)
```

In [195…
```
cmat = confusion_matrix(y_test, pred, labels = [0,1])
s = sns.heatmap(cmat,annot = True, fmt = 'g',xticklabels= [0,1],yticklabels= [0,1])
plt.xlabel("Predicted")
plt.xlabel("Actual")
plt.title("Confusion Matrix")
plt.show()
acc = cmat.trace()/cmat.sum()
print('Accuracy: {0:5.2f}%'.format(acc*100))
```



Accuracy: 83.12%

In [ ]: