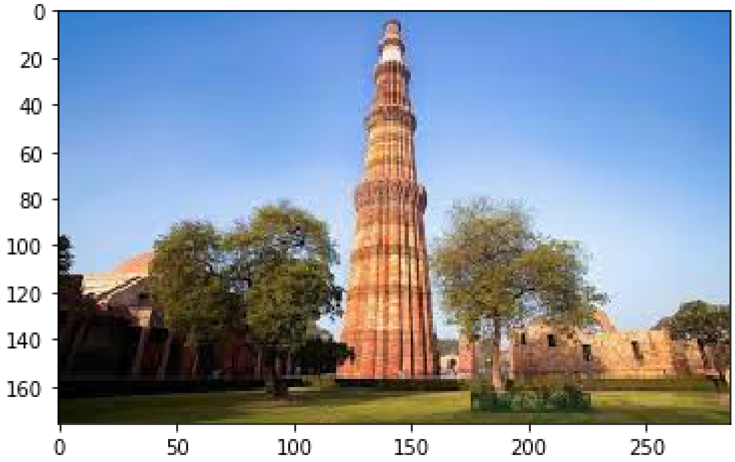
In [15]:

```
import numpy as np
import cv2
import imutils
import csv
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import math
import scipy.spatial
```

In [6]:

```
im=mpimg.imread('qutubminar.jpeg')
imgplot = plt.imshow(im)
```



In [19]:

```
feature=[]
feature=get_features(im)
```

In [22]:

```
def get_features(image):
    #convert from BGR to HSV color model
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    features = []
    (height, width) = image.shape[:2]
    (centerX, centerY) = (int(width * 0.5), int(height * 0.5))
    # top-left, top-right, bottom-right, bottom-left pieces of images
    pieces = [(0, centerX, 0, centerY), (centerX, width, 0, centerY), (
    (0, centerX, centerY, height)]
    # making an elliptical mask image center
    (x_axes, y_axes) = (int(width * 0.75) // 2, int(height * 0.75) // 2
    ellipse_Mask = np.zeros(image.shape[:2], dtype = "uint8")
    cv2.ellipse(ellipse_Mask, (centerX, centerY), (x_axes, y_axes), 0,
    for (sX, sX, sY, eY) in pieces:
        # construct a mask for each corner of the image, subtracting
        # the elliptical center from it
        corner = np.zeros(image.shape[:2], dtype = "uint8")
        cv2.rectangle(cornerMask, (sX, sY), (sX, eY), 255, -1)
        cornerMask = cv2.subtract(cornerMask, ellipse_Mask)
        # extract a color histogram from the image, then update the
        # feature vector
        hist = cv2.calcHist([image], [0, 1, 2], corner, [8,12,3],[0, 18
        hist = cv2.normalize(hist, hist).flatten()
        features.extend(hist)

    # extract a color histogram from the elliptical region and
    # update the feature vector
    hist = cv2.calcHist([image], [0, 1, 2], ellipse_Mask, [8,12,3],[0,
    hist = cv2.normalize(hist, hist).flatten()
    features.extend(hist)

    # return the feature vector
    return features
```

In [9]:

```
len(feature)
```

Out[9]:

```
0
```

In [17]:

```
def search(queryFeatures):
    # initialize our dictionary of results
    results = {}
    limit = 10
    # open the index file for reading
    with open('index.csv') as f:
    # initialize the CSV reader
        reader = csv.reader(f)

    # loop over the rows in the index
        for row in reader:
                # parse out the image ID and features, then compute the
                # chi-squared distance between the features in our inde
                # and our query features
            features = [float(x) for x in row[1:]]
            d =cos_distance(features, queryFeatures)

                # now that we have the distance between the two feature
                # vectors, we can udpate the results dictionary -- the
                # key is the current image ID in the index and the
                # value is the distance we just computed, representing
                # how 'similar' the image in the index is to our query
            results[row[0]] = d

            # close the reader
        f.close()

        # sort our results, so that the smaller distances (i.e. the
        # more relevant images are at the front of the list)
        results = sorted([(v, k) for (k, v) in results.items()])

        # return our (limited) results
        return results[:limit]
```

In [11]:

```
def euclidean_dist(vector_1, vector_2):
        #euclidean distance
        sum3=0
        for i in range(2,len(vector_1)):
            sum3=sum3+(math.sqrt(abs(vector_1[i]-vector_2[i])))
        #for (a, b) in zip(vector_1, vector_2)

        # return the euclidean distance
        return sum3
```

In [12]:

```
def chi2_distance(histA, histB, eps = 1e-10):
        # compute the chi-squared distance
        d = 0.5 * np.sum([((a - b) ** 2) / (a + b + eps)
            for (a, b) in zip(histA, histB)])

        # return the chi-squared distance
        return d
```

In [16]:

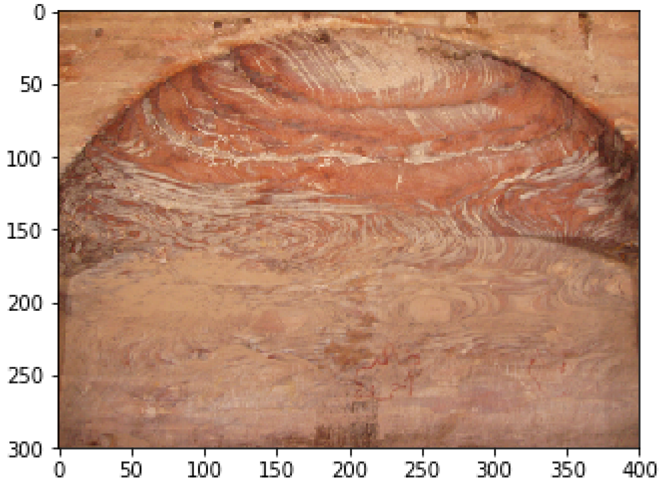
```
def cos_distance(histA,histB):
    v=np.asarray(histB)
    v1=np.asarray(histA)
    v = v.reshape(1, -1)
    v1=v1.reshape(1, -1)
    d=scipy.spatial.distance.cdist(v,v1,'cosine').reshape(-1)
    return d
```

In [20]:

```
result_set=search(feature)
```

In [21]:

```
for i in range(0,len(result_set)):
    img=mpimg.imread(result_set[i][1])
    imgplot = plt.imshow(img)
    plt.show()
```



In [ ]:

```

```

In [ ]:

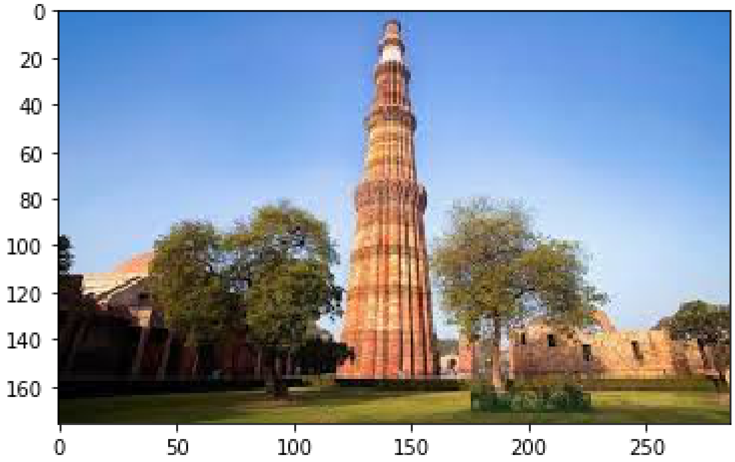
```

```

In [1]:

```
import numpy as np
import cv2
import imutils
import csv
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import math
import scipy.spatial
```

In [2]:

```
im=mpimg.imread('qutubminar.jpeg')
imgplot = plt.imshow(im)
```



In [7]:

```
feature=[]
feature=get_features(im)
```

In [6]:

```
def get_features(image):
    #convert from BGR to HSV color model
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    features = []
    (height, width) = image.shape[:2]
    (centerX, centerY) = (int(width * 0.5), int(height * 0.5))
    # top-left, top-right, bottom-right, bottom-left pieces of images
    pieces = [(0, centerX, 0, centerY), (centerX, width, 0, centerY), (
    (0, centerX, centerY, height)]
    # making an elliptical mask image center
    (x_axes, y_axes) = (int(width * 0.75) // 2, int(height * 0.75) // 2
    ellipse_Mask = np.zeros(image.shape[:2], dtype = "uint8")
    cornerMask=cv2.ellipse(ellipse_Mask, (centerX, centerY), (x_axes, y
    for (sX, sX, sY, eY) in pieces:
        # construct a mask for each corner of the image, subtracting
        # the elliptical center from it
        corner = np.zeros(image.shape[:2], dtype = "uint8")
        cv2.rectangle(cornerMask, (sX, sY), (sX, eY), 255, -1)
        cornerMask = cv2.subtract(cornerMask, ellipse_Mask)
        # extract a color histogram from the image, then update the
        # feature vector
        hist = cv2.calcHist([image], [0, 1, 2], corner, [8,12,3],[0, 18
        hist = cv2.normalize(hist, hist).flatten()
        features.extend(hist)

    # extract a color histogram from the elliptical region and
    # update the feature vector
    hist = cv2.calcHist([image], [0, 1, 2], ellipse_Mask, [8,12,3],[0,
    hist = cv2.normalize(hist, hist).flatten()
    features.extend(hist)

    # return the feature vector
    return features
```

In [8]:

```
len(feature)
```

Out[8]:

```
1440
```

In [9]:

```
def search(queryFeatures):
    # initialize our dictionary of results
    results = {}
    limit = 10
    # open the index file for reading
    with open('index.csv') as f:
    # initialize the CSV reader
        reader = csv.reader(f)

    # loop over the rows in the index
        for row in reader:
                # parse out the image ID and features, then compute the
                # chi-squared distance between the features in our inde
                # and our query features
            features = [float(x) for x in row[1:]]
            d =chi2_distance(features, queryFeatures)

                # now that we have the distance between the two feature
                # vectors, we can udpate the results dictionary -- the
                # key is the current image ID in the index and the
                # value is the distance we just computed, representing
                # how 'similar' the image in the index is to our query
            results[row[0]] = d

            # close the reader
        f.close()

        # sort our results, so that the smaller distances (i.e. the
        # more relevant images are at the front of the list)
        results = sorted([(v, k) for (k, v) in results.items()])

        # return our (limited) results
        return results[:limit]
```

In [10]:

```
def euclidean_dist(vector_1, vector_2):
        #euclidean distance
        sum3=0
        for i in range(2,len(vector_1)):
            sum3=sum3+(math.sqrt(abs(vector_1[i]-vector_2[i])))
        #for (a, b) in zip(vector_1, vector_2)

        # return the euclidean distance
        return sum3
```

In [11]:

```
def chi2_distance(histA, histB, eps = 1e-10):
        # compute the chi-squared distance
        d = 0.5 * np.sum([((a - b) ** 2) / (a + b + eps)
            for (a, b) in zip(histA, histB)])

        # return the chi-squared distance
        return d
```

In [12]:

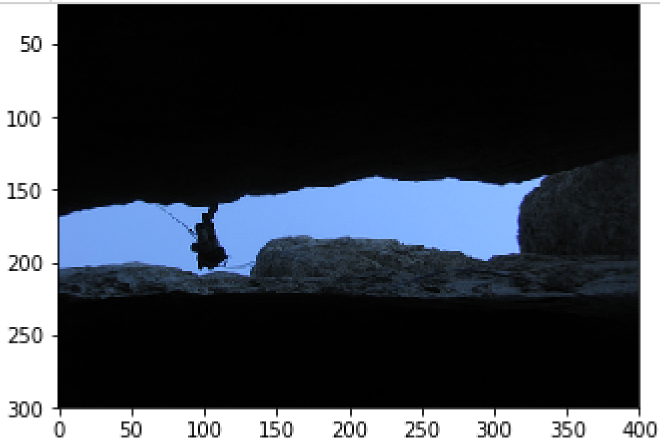
```python
def cos_distance(histA,histB):
    v=np.asarray(histB)
    v1=np.asarray(histA)
    v = v.reshape(1, -1)
    v1=v1.reshape(1, -1)
    d=scipy.spatial.distance.cdist(v,v1,'cosine').reshape(-1)
    return d
```

In [13]:

```python
result_set=search(feature)
```

In [14]:

```python
for i in range(0,len(result_set)):
    img=mpimg.imread(result_set[i][1])
    imgplot = plt.imshow(img)
    plt.show()
```



In [ ]:

```

```

In [ ]:

```

```