HW1 Q2 K-means Arjun Singh

May 24, 2020

K-Means algorithm. Please note that I have reused some of my old code from CSE 6040.

[1]:
```
import numpy as np
import time
```

[32]:
```
#   Randomly selects k points from the input image
def init_centers(X, k):

    centers = np.random.choice(len(X),k,replace=False)
#     centers = np.random.choice(len(X),1,replace=False)
#     return X[np.repeat(centers,k),:]
    return X[centers,:]
```

[55]:
```
# Compute Euclidean distance
def compute_d2(X, centers):

    m = len(X)
    s=len(centers)

    S = np.empty((m,s))
    for i in range(m):
        S[i,:] = np.linalg.norm(X[i,:]-centers,ord=2,axis=1)**2

    return S
```

[56]:
```
def assign_cluster_labels(S):
    return np.argmin(S,axis=1)
```

[57]:
```
def update_centers(X, y):
    # X[:m, :d] == m points, each of dimension d
    # y[:m] == cluster labels
    m, d = X.shape
    k = max(y) + 1
    assert m == len(y)
    assert (min(y) >= 0)

    centers = np.empty((k, d))
    for j in range(k):
```

```
        # Compute the new center of cluster j,
        # i.e., centers[j, :d].
        centers[j,:] = np.mean(X[j==y],axis=0)
    return centers
```

[58]:
```
# Calculate the within clusters sum of squares
def WCSS(S):
    return np.sum(np.amin(S,axis=1))
```

[23]:
```
# The algorithm has converged if there is no change in the cluster centers in␣
 ↪the next iteration
def has_converged(old_centers, centers):
    return set([tuple(x) for x in old_centers]) == set([tuple(x) for x in␣
 ↪centers])
```

[24]:
```
def kmeans(X, k,starting_centers=None,max_steps=np.inf):
    start = time.time()

    if starting_centers is None:
        centers = init_centers(X, k)
    else:
        centers = starting_centers

    converged = False
    labels = np.zeros(len(X))
    i = 1
    while (not converged) and (i <= max_steps):
        old_centers = centers
        S = compute_d2(X, old_centers)
        labels = assign_cluster_labels(S)
        centers = update_centers(X, labels)
        wc = WCSS(S)

        converged = has_converged(old_centers,centers)
        print ("iteration", i, "WCSS = ", WCSS (S))
        i += 1
    stop = time.time()
    print("Time taken = {:.2f} seconds".format(stop-start))
    return labels
```

[25]:
```
from matplotlib.pyplot import imshow
%matplotlib inline
def display_image(arr):
    """
    display the image
    input : 3 dimensional array
    """
```

```
    arr = arr.astype(dtype='uint8')
    img = Image.fromarray(arr, 'RGB')
    imshow(np.asarray(img))
```

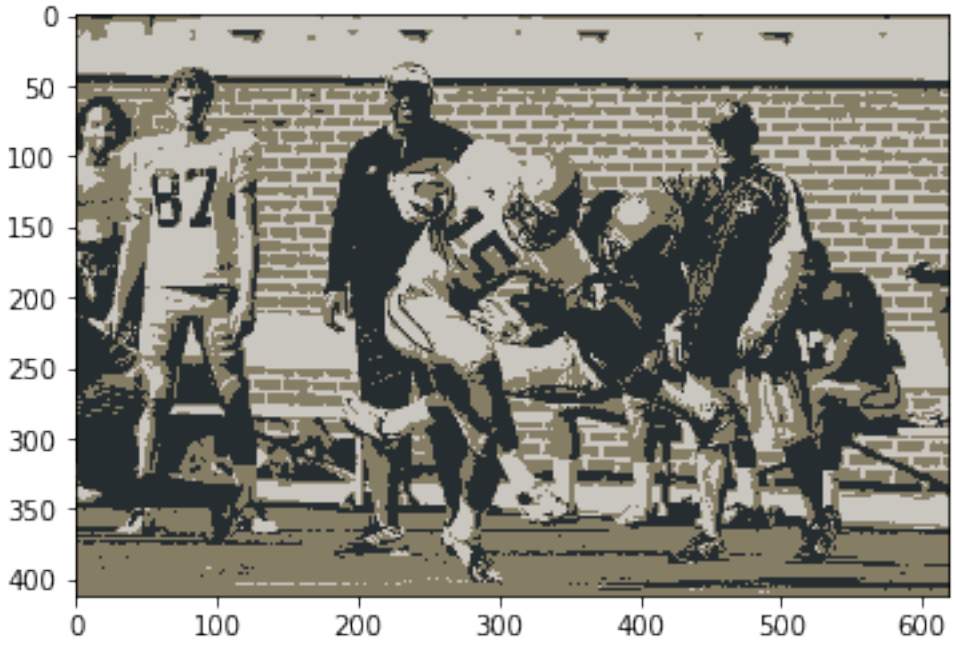
[26]:
```
from PIL import Image
def read_img(path):
    """
    Read image and store it as an array, given the image path.
    Returns the 3 dimensional image array.
    """
    img = Image.open(path)
    img_arr = np.array(img, dtype='int32')
    img.close()
    return img_arr
```

[27]:
```
def runKMeansAlgorithm(pixels,k):
    print("Running k-means algorithm...")
#   Read image
    image = read_img(pixels)
    r, c, l = image.shape
#   Flatten image
    img_reshaped = np.reshape(image, (r*c, l), order="C")
#   Run k-means algorithm
    labels = kmeans(img_reshaped, k, starting_centers=None)
    ind = np.column_stack((img_reshaped, labels))
    centers = {}
    for i in set(labels):
        c = ind[ind[:,3] == i].mean(axis=0)
        centers[i] = c[:3]

    img_clustered = np.array([centers[i] for i in labels])
    r, c, l = image.shape
    img_disp = np.reshape(img_clustered, (r, c, l), order="C")

    print('Image with k = ' + str(k) + " clusters...")
    display_image(img_disp)
    print("The labels are: {}".format(labels.T))
    print("The cluster centers are {}".format(centers))
    return labels,centers
```

[36]:
```
lab, cen = runKMeansAlgorithm("data/football.bmp",3)
```

```
Running k-means algorithm…
iteration 1 WCSS =  2456316508.0
iteration 2 WCSS =  954303493.6362097
iteration 3 WCSS =  671747803.6800143
iteration 4 WCSS =  639939317.5572674
```

```
iteration 5 WCSS =  635886999.0335318
iteration 6 WCSS =  634830313.722801
iteration 7 WCSS =  634448944.3235377
iteration 8 WCSS =  634294560.9102862
iteration 9 WCSS =  634226091.4041917
iteration 10 WCSS =  634195392.6893654
iteration 11 WCSS =  634182298.9958154
iteration 12 WCSS =  634178085.6680316
iteration 13 WCSS =  634175679.265678
iteration 14 WCSS =  634174624.9946772
iteration 15 WCSS =  634174059.4714675
iteration 16 WCSS =  634173882.9721487
iteration 17 WCSS =  634173783.4825951
iteration 18 WCSS =  634173756.5604436
iteration 19 WCSS =  634173755.1569637
Time taken = 133.51 seconds
Image with k = 3 clusters…
The labels are: [2 2 2 … 0 0 0]
The cluster centers are {0: array([134.23584777, 125.34568766, 101.93988768]),
1: array([202.52108949, 198.84707504, 192.62337998]), 2: array([38.7890798 ,
45.35163548, 48.17869416])}
```



[37]:
```
lab, cen = runKMeansAlgorithm("data/football.bmp",16)
```

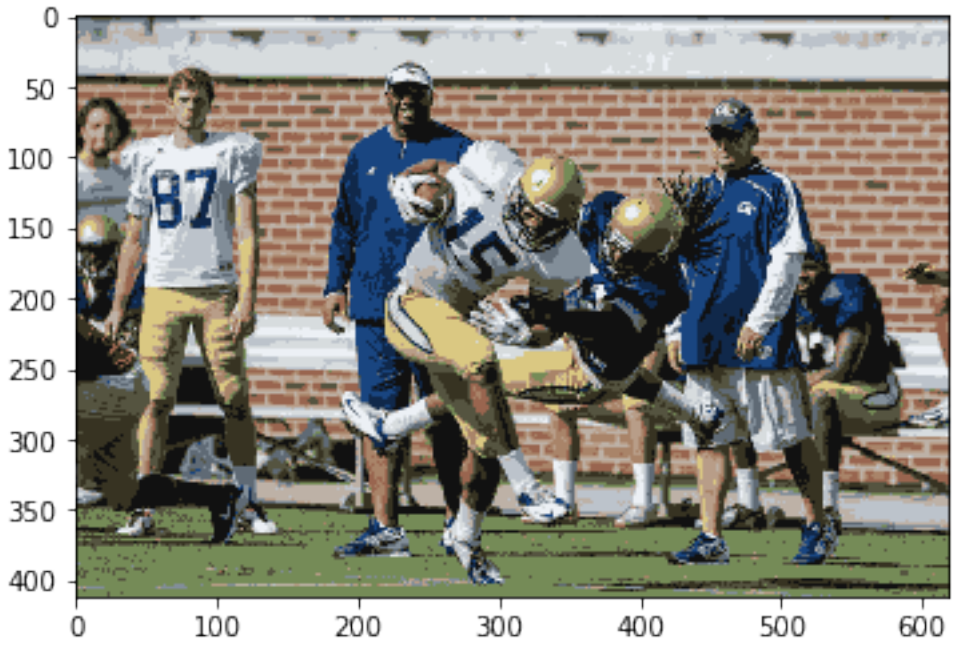
```
Running k-means algorithm…
iteration 1 WCSS =  389599008.0
```

```
iteration 2 WCSS =  194633966.26500764
iteration 3 WCSS =  158409209.34289968
iteration 4 WCSS =  148831373.6483631
iteration 5 WCSS =  143824610.22942817
iteration 6 WCSS =  139707579.16874346
iteration 7 WCSS =  137296156.4037293
iteration 8 WCSS =  135397899.4588457
iteration 9 WCSS =  133925560.75748907
iteration 10 WCSS =  132690377.51961173
iteration 11 WCSS =  131667817.91402242
iteration 12 WCSS =  130892740.08558576
iteration 13 WCSS =  130364212.98891617
iteration 14 WCSS =  129992012.59922427
iteration 15 WCSS =  129693213.8722543
iteration 16 WCSS =  129440147.52493048
iteration 17 WCSS =  129184506.98292685
iteration 18 WCSS =  128887669.48954152
iteration 19 WCSS =  128557986.11477509
iteration 20 WCSS =  128212061.16508226
iteration 21 WCSS =  127908373.72405538
iteration 22 WCSS =  127675522.41787711
iteration 23 WCSS =  127514064.77921885
iteration 24 WCSS =  127381072.11134267
iteration 25 WCSS =  127296917.37258105
iteration 26 WCSS =  127219626.15762621
iteration 27 WCSS =  127135595.99641562
iteration 28 WCSS =  127084561.22413416
iteration 29 WCSS =  127031114.71142764
iteration 30 WCSS =  126965027.98333591
iteration 31 WCSS =  126903670.89468691
iteration 32 WCSS =  126833696.83605571
iteration 33 WCSS =  126778082.5017755
iteration 34 WCSS =  126713484.00201091
iteration 35 WCSS =  126660603.66072176
iteration 36 WCSS =  126611707.37352398
iteration 37 WCSS =  126570049.28539969
iteration 38 WCSS =  126543355.6095167
iteration 39 WCSS =  126516175.03414136
iteration 40 WCSS =  126490316.327382
iteration 41 WCSS =  126469858.9570937
iteration 42 WCSS =  126447819.61642782
iteration 43 WCSS =  126426702.5349611
iteration 44 WCSS =  126404688.88800836
iteration 45 WCSS =  126381547.31152752
iteration 46 WCSS =  126355655.47235285
iteration 47 WCSS =  126325482.90294963
iteration 48 WCSS =  126291132.93700808
iteration 49 WCSS =  126249281.98181044
```

```
iteration 50 WCSS =  126203599.13343744
iteration 51 WCSS =  126146278.15400615
iteration 52 WCSS =  126067165.09515291
iteration 53 WCSS =  125947765.59821501
iteration 54 WCSS =  125801496.48122527
iteration 55 WCSS =  125618602.93741326
iteration 56 WCSS =  125398772.76508854
iteration 57 WCSS =  125168474.76176748
iteration 58 WCSS =  124950028.84253912
iteration 59 WCSS =  124713953.23097008
iteration 60 WCSS =  124428617.59272456
iteration 61 WCSS =  124121577.87629376
iteration 62 WCSS =  123850386.9459919
iteration 63 WCSS =  123617743.04269174
iteration 64 WCSS =  123429225.64908963
iteration 65 WCSS =  123278113.47015174
iteration 66 WCSS =  123151183.77912122
iteration 67 WCSS =  123039877.10813873
iteration 68 WCSS =  122947117.21163964
iteration 69 WCSS =  122872878.66243537
iteration 70 WCSS =  122805320.62898089
iteration 71 WCSS =  122743479.75850767
iteration 72 WCSS =  122685088.12013635
iteration 73 WCSS =  122635958.78536704
iteration 74 WCSS =  122596086.12046702
iteration 75 WCSS =  122561986.22790265
iteration 76 WCSS =  122535067.16891102
iteration 77 WCSS =  122509628.66869356
iteration 78 WCSS =  122486993.20189187
iteration 79 WCSS =  122467116.37805513
iteration 80 WCSS =  122446064.64995241
iteration 81 WCSS =  122427774.84981969
iteration 82 WCSS =  122415317.02470523
iteration 83 WCSS =  122402974.2085328
iteration 84 WCSS =  122390120.51490752
iteration 85 WCSS =  122378540.14667523
iteration 86 WCSS =  122371670.88245495
iteration 87 WCSS =  122366985.29189879
iteration 88 WCSS =  122363660.70652373
iteration 89 WCSS =  122358712.20760457
iteration 90 WCSS =  122352572.05131795
iteration 91 WCSS =  122345898.35536122
iteration 92 WCSS =  122341346.20573317
iteration 93 WCSS =  122338603.31101723
iteration 94 WCSS =  122336909.27342421
iteration 95 WCSS =  122335862.93781473
iteration 96 WCSS =  122335002.74908623
iteration 97 WCSS =  122334241.98644127
```

```
iteration 98 WCSS =  122332762.26002996
iteration 99 WCSS =  122331049.79412797
iteration 100 WCSS =  122329643.48998876
iteration 101 WCSS =  122327973.11487384
iteration 102 WCSS =  122325495.81644262
iteration 103 WCSS =  122323208.82716674
iteration 104 WCSS =  122321957.89813724
iteration 105 WCSS =  122321089.87948734
iteration 106 WCSS =  122319239.34875216
iteration 107 WCSS =  122317379.33636372
iteration 108 WCSS =  122316142.20912726
iteration 109 WCSS =  122315428.50033154
iteration 110 WCSS =  122315026.59455879
iteration 111 WCSS =  122314662.79439995
iteration 112 WCSS =  122314310.05917099
iteration 113 WCSS =  122314042.53917743
iteration 114 WCSS =  122313775.93435095
iteration 115 WCSS =  122313582.278637
iteration 116 WCSS =  122313464.33245645
iteration 117 WCSS =  122313011.4310619
iteration 118 WCSS =  122312377.52659644
iteration 119 WCSS =  122312000.4171061
iteration 120 WCSS =  122311730.82577868
iteration 121 WCSS =  122311498.76539102
iteration 122 WCSS =  122311168.37947495
iteration 123 WCSS =  122310871.68119371
iteration 124 WCSS =  122310591.64208212
iteration 125 WCSS =  122310375.34639943
iteration 126 WCSS =  122310279.15037306
iteration 127 WCSS =  122310225.40522182
iteration 128 WCSS =  122310183.55788857
iteration 129 WCSS =  122310158.0667483
iteration 130 WCSS =  122310129.83383945
iteration 131 WCSS =  122308552.96051888
iteration 132 WCSS =  122308233.74553815
iteration 133 WCSS =  122308197.40648106
iteration 134 WCSS =  122308176.44870742
iteration 135 WCSS =  122308147.07327214
iteration 136 WCSS =  122308144.55726838
iteration 137 WCSS =  122308141.03021088
iteration 138 WCSS =  122308140.55316299
Time taken = 977.91 seconds
Image with k = 16 clusters…
The labels are: [5 5 5 … 1 1 1]
The cluster centers are {0: array([188.49935166, 195.5687889 , 203.2629668 ]),
1: array([117.8208072 , 139.48794729,  87.18471614]), 2: array([ 25.10745839,
69.52558044, 128.11177317]), 3: array([55.37723925, 51.19669001, 34.75188688]),
4: array([184.08713364, 157.50175399, 136.74663347]), 5: array([85.42213893,
```

```
84.03601253, 64.75678497]), 6: array([17.83079513, 17.89143797, 17.12841091]),
7: array([236.34535561, 240.43735763, 245.13642116]), 8: array([153.1981332 ,
101.26696444,  79.546756  ]), 9: array([16.31240223, 40.05441341, 74.45296089]),
10: array([189.47646148, 177.33742699, 166.95047676]), 11: array([215.86818003,
220.53727157, 222.15947237]), 12: array([130.27638546, 142.86648429,
153.58656901]), 13: array([218.22310208, 200.73541057, 130.67877432]), 14:
array([ 95.24515007, 106.94637628, 117.70516105]), 15: array([169.78753541,
132.18570151, 105.5514388 ])}
```



[38]:
```
lab, cen = runKMeansAlgorithm("data/football.bmp",32)
```

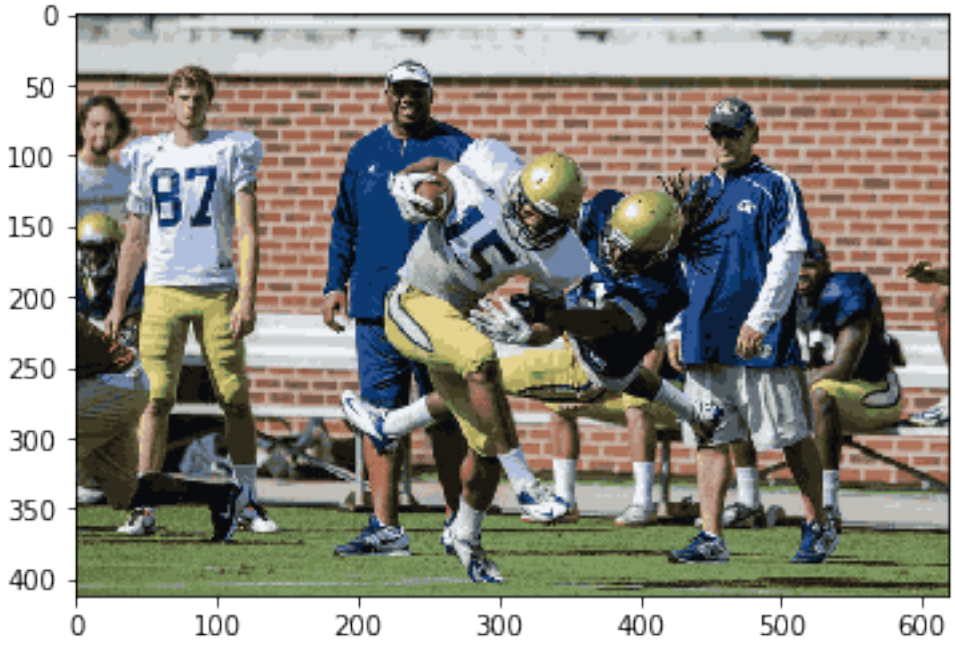
```
Running k-means algorithm…
iteration 1 WCSS =  154156513.0
iteration 2 WCSS =  90854426.99404576
iteration 3 WCSS =  82374936.18956882
iteration 4 WCSS =  78979555.4186059
iteration 5 WCSS =  76874969.7167158
iteration 6 WCSS =  75376297.9050194
iteration 7 WCSS =  74181937.53857802
iteration 8 WCSS =  73510089.71580961
iteration 9 WCSS =  73042821.19014058
iteration 10 WCSS =  72611144.12092553
iteration 11 WCSS =  72173924.06588945
iteration 12 WCSS =  71681196.24227656
iteration 13 WCSS =  71134192.82620926
iteration 14 WCSS =  70540873.17103964
```

```
iteration 15 WCSS =  69924895.59256424
iteration 16 WCSS =  69313949.72028396
iteration 17 WCSS =  68802180.49051738
iteration 18 WCSS =  68392598.33023295
iteration 19 WCSS =  68090924.64142045
iteration 20 WCSS =  67890058.1555307
iteration 21 WCSS =  67739501.48248678
iteration 22 WCSS =  67606630.3922924
iteration 23 WCSS =  67473454.12835175
iteration 24 WCSS =  67355732.49085784
iteration 25 WCSS =  67255161.68398519
iteration 26 WCSS =  67191316.78659722
iteration 27 WCSS =  67139978.00513403
iteration 28 WCSS =  67090993.50562282
iteration 29 WCSS =  67051415.1236087
iteration 30 WCSS =  67004459.00868036
iteration 31 WCSS =  66960393.448468596
iteration 32 WCSS =  66923387.652685605
iteration 33 WCSS =  66886740.578007676
iteration 34 WCSS =  66852165.68997419
iteration 35 WCSS =  66818579.16168487
iteration 36 WCSS =  66776739.065597385
iteration 37 WCSS =  66724878.49203261
iteration 38 WCSS =  66667391.04308212
iteration 39 WCSS =  66612905.44686758
iteration 40 WCSS =  66558704.06604199
iteration 41 WCSS =  66512063.22512857
iteration 42 WCSS =  66461844.02640541
iteration 43 WCSS =  66416672.93931889
iteration 44 WCSS =  66383996.182326905
iteration 45 WCSS =  66357117.885337256
iteration 46 WCSS =  66331327.83996085
iteration 47 WCSS =  66309972.26288962
iteration 48 WCSS =  66290467.90574199
iteration 49 WCSS =  66273431.62863433
iteration 50 WCSS =  66257536.00882024
iteration 51 WCSS =  66240433.76244594
iteration 52 WCSS =  66219918.11447064
iteration 53 WCSS =  66197567.31849523
iteration 54 WCSS =  66174297.33178489
iteration 55 WCSS =  66148462.15137137
iteration 56 WCSS =  66120285.63237766
iteration 57 WCSS =  66091428.925723195
iteration 58 WCSS =  66063793.93462781
iteration 59 WCSS =  66039291.65311639
iteration 60 WCSS =  66020576.33910455
iteration 61 WCSS =  66005384.563282736
iteration 62 WCSS =  65993470.60843236
```

```
iteration 63 WCSS =  65984993.968146615
iteration 64 WCSS =  65977775.38169028
iteration 65 WCSS =  65971341.89105098
iteration 66 WCSS =  65964897.213070914
iteration 67 WCSS =  65958693.217298485
iteration 68 WCSS =  65954604.293055214
iteration 69 WCSS =  65950826.3306454
iteration 70 WCSS =  65948188.39329451
iteration 71 WCSS =  65945365.36959097
iteration 72 WCSS =  65942145.552824594
iteration 73 WCSS =  65939293.8508109
iteration 74 WCSS =  65936503.16170342
iteration 75 WCSS =  65934431.868236944
iteration 76 WCSS =  65933017.755912535
iteration 77 WCSS =  65931591.60981239
iteration 78 WCSS =  65930179.809461065
iteration 79 WCSS =  65928444.1488754
iteration 80 WCSS =  65926988.283575125
iteration 81 WCSS =  65925476.44836222
iteration 82 WCSS =  65924130.256465636
iteration 83 WCSS =  65922833.017187074
iteration 84 WCSS =  65921666.39844116
iteration 85 WCSS =  65920353.24527504
iteration 86 WCSS =  65918798.59710905
iteration 87 WCSS =  65917685.66757907
iteration 88 WCSS =  65916766.63037999
iteration 89 WCSS =  65916037.341308005
iteration 90 WCSS =  65915276.57709618
iteration 91 WCSS =  65914429.30425806
iteration 92 WCSS =  65913942.53997429
iteration 93 WCSS =  65913646.910160765
iteration 94 WCSS =  65913327.305902086
iteration 95 WCSS =  65912936.29789154
iteration 96 WCSS =  65912669.35526211
iteration 97 WCSS =  65912495.03862722
iteration 98 WCSS =  65912367.35838308
iteration 99 WCSS =  65912226.32508523
iteration 100 WCSS =  65912109.62433027
iteration 101 WCSS =  65911981.971656516
iteration 102 WCSS =  65911754.81439332
iteration 103 WCSS =  65911620.67678566
iteration 104 WCSS =  65911509.611392036
iteration 105 WCSS =  65911453.5028878
iteration 106 WCSS =  65911380.97904354
iteration 107 WCSS =  65911338.20146704
iteration 108 WCSS =  65911317.31561838
iteration 109 WCSS =  65911294.13023815
iteration 110 WCSS =  65911241.64068819
```

```
iteration 111 WCSS =  65911179.67890533
iteration 112 WCSS =  65911136.58227505
iteration 113 WCSS =  65911110.88510049
iteration 114 WCSS =  65911090.30270436
iteration 115 WCSS =  65911073.654683575
iteration 116 WCSS =  65911069.30501881
iteration 117 WCSS =  65911066.52864003
iteration 118 WCSS =  65911058.1734749
iteration 119 WCSS =  65911034.697925135
iteration 120 WCSS =  65911008.16123636
iteration 121 WCSS =  65910963.60735541
iteration 122 WCSS =  65910903.772770904
iteration 123 WCSS =  65910863.03522626
iteration 124 WCSS =  65910845.067064956
iteration 125 WCSS =  65910821.73000511
iteration 126 WCSS =  65910765.71878215
iteration 127 WCSS =  65910741.73739506
iteration 128 WCSS =  65910714.2128681
iteration 129 WCSS =  65910667.364258714
iteration 130 WCSS =  65910645.97710661
iteration 131 WCSS =  65910644.67183107
iteration 132 WCSS =  65910638.85006445
iteration 133 WCSS =  65910624.275976114
iteration 134 WCSS =  65910619.72531885
iteration 135 WCSS =  65910618.370981656
iteration 136 WCSS =  65910617.67236624
iteration 137 WCSS =  65910616.73920506
iteration 138 WCSS =  65910616.04407202
iteration 139 WCSS =  65910615.53114029
iteration 140 WCSS =  65910615.309055164
iteration 141 WCSS =  65910615.21597049
Time taken = 1025.84 seconds
Image with k = 32 clusters…
The labels are: [31 31  5 …  9 20 20]
The cluster centers are {0: array([158.78160667,  99.65803336,  76.5679324 ]),
1: array([174.68212603, 115.90650766,  91.59231449]), 2: array([239.01673573,
242.75183296, 247.21883966]), 3: array([186.32316412, 158.03742602,
141.36934234]), 4: array([46.14246052, 42.86300538, 28.69772688]), 5:
array([55.35414145, 64.04634273, 68.70776414]), 6: array([13.47059673,
32.10981072, 61.07311082]), 7: array([85.25054693, 81.00885509, 56.88352953]),
8: array([ 42.47548121,  81.38817599, 142.42163153]), 9: array([127.76199978,
152.31770314,  97.35417359]), 10: array([176.65245255, 191.83657875,
222.98940104]), 11: array([122.8435745 ,  90.90303931,  74.11446977]), 12:
array([100.62787424, 117.75113405,  69.49053652]), 13: array([ 12.06051948,
57.78662338, 112.86948052]), 14: array([109.92644295, 132.01369128,
167.12805369]), 15: array([191.69427896, 190.91432624, 185.39725768]), 16:
array([202.93570275, 190.7098546 , 115.18804523]), 17: array([229.66644137,
211.26225076, 144.17269348]), 18: array([176.54366094, 138.52563634,
```

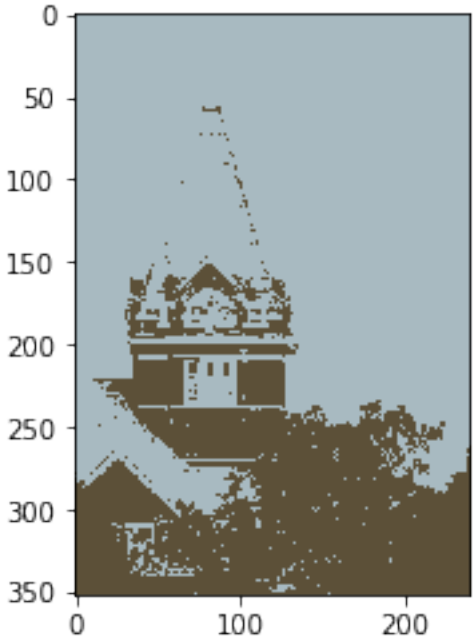
```
119.05516989]), 19: array([67.62414898, 60.51281538, 37.51621946]), 20:
array([114.66434635, 137.0098472 ,  83.27410866]), 21: array([220.20979827,
224.79688761, 226.64616715]), 22: array([173.10397295, 163.40617075,  95.4103973
]), 23: array([103.39973615, 114.95705072, 125.09073585]), 24:
array([204.23231629, 209.35333047, 210.16347228]), 25: array([ 8.43815496,
10.64955854, 13.57909068]), 26: array([27.67181202, 24.02332562, 17.54746591]),
27: array([140.92210682, 117.52670623, 102.34254451]), 28: array([197.51942974,
174.29344196, 160.16211813]), 29: array([160.68984363, 170.2398664 ,
173.94899044]), 30: array([141.61348454, 147.20149138, 140.97296877]), 31:
array([81.60524017, 91.83435226, 95.6209607 ])}
```



[39]:
```
lab, cen = runKMeansAlgorithm("data/TechTower.jpg",2)
```

```
Running k-means algorithm…
iteration 1 WCSS =  404111557.0
iteration 2 WCSS =  185068009.59071818
iteration 3 WCSS =  170311286.1525032
iteration 4 WCSS =  169676642.10593435
iteration 5 WCSS =  169657307.86373463
iteration 6 WCSS =  169656916.80336154
iteration 7 WCSS =  169656891.6555998
iteration 8 WCSS =  169656890.81534788
Time taken = 18.93 seconds
Image with k = 2 clusters…
The labels are: [1 1 1 … 0 0 0]
The cluster centers are {0: array([92.84412979, 80.94779891, 56.73093098]), 1:
```

```
array([168.57808098, 186.45381739, 193.0552757 ])}
```

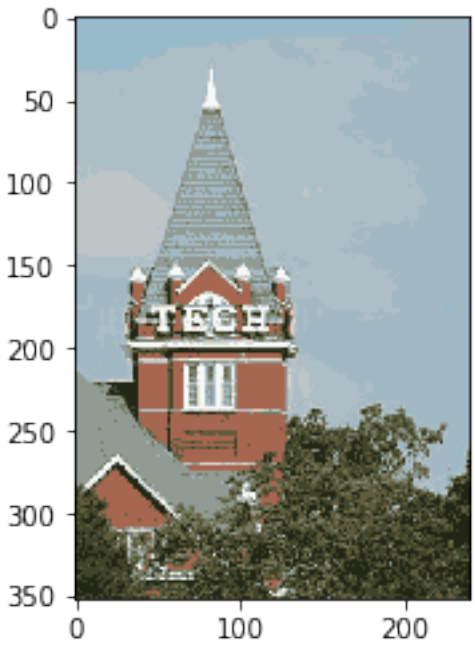


[40]:
```
lab, cen = runKMeansAlgorithm("data/TechTower.jpg",12)
```

```
Running k-means algorithm…
iteration 1 WCSS =  78081187.0
iteration 2 WCSS =  38936409.903284445
iteration 3 WCSS =  31357584.93121798
iteration 4 WCSS =  29227598.9107575
iteration 5 WCSS =  28650936.17738529
iteration 6 WCSS =  28411226.065431207
iteration 7 WCSS =  28192688.736099314
iteration 8 WCSS =  27965000.435471594
iteration 9 WCSS =  27583585.606403805
iteration 10 WCSS =  26785410.072711565
iteration 11 WCSS =  26346085.22983884
iteration 12 WCSS =  26048507.612918265
iteration 13 WCSS =  25797227.60573025
iteration 14 WCSS =  25566364.945790816
iteration 15 WCSS =  25338954.926837184
iteration 16 WCSS =  25118219.123322424
iteration 17 WCSS =  24830252.30974612
iteration 18 WCSS =  24360939.59759022
iteration 19 WCSS =  23748258.6143158
iteration 20 WCSS =  23169536.275939815
iteration 21 WCSS =  22679166.139020912
```

```
iteration 22 WCSS =  22317896.191743504
iteration 23 WCSS =  22025442.517010275
iteration 24 WCSS =  21802724.609557293
iteration 25 WCSS =  21624312.162718233
iteration 26 WCSS =  21480177.61772822
iteration 27 WCSS =  21363760.94427106
iteration 28 WCSS =  21276912.28350205
iteration 29 WCSS =  21196653.6173141
iteration 30 WCSS =  21124808.580648296
iteration 31 WCSS =  21065369.621132564
iteration 32 WCSS =  21015204.797175527
iteration 33 WCSS =  20970273.4595458
iteration 34 WCSS =  20931816.09624045
iteration 35 WCSS =  20899758.778062407
iteration 36 WCSS =  20875077.24558523
iteration 37 WCSS =  20855216.874085985
iteration 38 WCSS =  20838273.204672683
iteration 39 WCSS =  20821200.45262719
iteration 40 WCSS =  20806930.92696262
iteration 41 WCSS =  20794085.365136594
iteration 42 WCSS =  20774897.274009753
iteration 43 WCSS =  20763147.170220234
iteration 44 WCSS =  20748979.812213104
iteration 45 WCSS =  20740097.265195515
iteration 46 WCSS =  20726748.7160254
iteration 47 WCSS =  20715893.216240577
iteration 48 WCSS =  20705705.997667044
iteration 49 WCSS =  20692077.80484623
iteration 50 WCSS =  20682016.024851672
iteration 51 WCSS =  20667964.003256697
iteration 52 WCSS =  20655439.116513554
iteration 53 WCSS =  20643727.054703187
iteration 54 WCSS =  20633186.51691845
iteration 55 WCSS =  20620871.853051785
iteration 56 WCSS =  20606556.20117656
iteration 57 WCSS =  20593357.742382225
iteration 58 WCSS =  20579793.604372565
iteration 59 WCSS =  20568208.0191338
iteration 60 WCSS =  20556219.009414617
iteration 61 WCSS =  20545535.895636324
iteration 62 WCSS =  20538471.687272158
iteration 63 WCSS =  20530973.459324382
iteration 64 WCSS =  20523356.111371353
iteration 65 WCSS =  20517006.505791984
iteration 66 WCSS =  20510681.263629843
iteration 67 WCSS =  20506576.851588584
iteration 68 WCSS =  20502768.098947916
iteration 69 WCSS =  20500118.385519367
```

```
iteration 70 WCSS =  20497376.686463825
iteration 71 WCSS =  20495799.64377183
iteration 72 WCSS =  20494220.044422064
iteration 73 WCSS =  20492874.263305224
iteration 74 WCSS =  20491847.578377612
iteration 75 WCSS =  20490746.616709426
iteration 76 WCSS =  20490197.50945997
iteration 77 WCSS =  20489837.242113557
iteration 78 WCSS =  20489570.67735978
iteration 79 WCSS =  20489210.12731411
iteration 80 WCSS =  20488900.940064684
iteration 81 WCSS =  20488739.237768378
iteration 82 WCSS =  20488630.46174035
iteration 83 WCSS =  20488589.671364233
iteration 84 WCSS =  20488581.27814218
iteration 85 WCSS =  20488572.305760473
iteration 86 WCSS =  20488553.176570028
iteration 87 WCSS =  20488508.172715124
iteration 88 WCSS =  20488428.128104635
iteration 89 WCSS =  20488366.354039375
iteration 90 WCSS =  20488291.20517655
iteration 91 WCSS =  20488260.062870495
iteration 92 WCSS =  20488240.21080901
iteration 93 WCSS =  20488216.908208635
iteration 94 WCSS =  20488209.89345389
iteration 95 WCSS =  20488204.266885314
iteration 96 WCSS =  20488183.404505007
iteration 97 WCSS =  20488144.810690735
iteration 98 WCSS =  20488132.606662843
iteration 99 WCSS =  20488127.732815232
iteration 100 WCSS =  20488123.090184085
iteration 101 WCSS =  20488122.38633112
iteration 102 WCSS =  20488122.02670927
Time taken = 238.52 seconds
Image with k = 12 clusters…
The labels are: [1 1 1 … 2 2 2]
The cluster centers are {0: array([63.64358452, 68.14533023, 42.10503346]), 1:
array([151.77939198, 187.2055421 , 203.47807372]), 2: array([34.08600237,
38.20897588, 21.03459866]), 3: array([118.91362764, 123.92098528,
100.57165707]), 4: array([167.08229295, 102.05820292,  79.61790047]), 5:
array([173.130089  , 190.73404658, 199.55259421]), 6: array([145.54917355,
155.75096419, 148.69669421]), 7: array([96.03782121, 93.43612016, 62.94299674]),
8: array([162.78061891, 185.26378981, 195.52436451]), 9: array([222.50398936,
216.93484043, 204.95212766]), 10: array([189.8633851 , 197.95051329,
199.44827586]), 11: array([247.74863719, 249.86977589, 248.30284676])}
```



[54]:
```
lab, cen = runKMeansAlgorithm("data/beach.bmp",32)
```

```
Running k-means algorithm…
iteration 1 WCSS =  45019968.0
iteration 2 WCSS =  27767575.689218067
iteration 3 WCSS =  23898738.075360753
iteration 4 WCSS =  22352898.928599153
iteration 5 WCSS =  21484255.380953792
iteration 6 WCSS =  20815592.421394795
iteration 7 WCSS =  20220248.45025015
iteration 8 WCSS =  19626269.678854804
iteration 9 WCSS =  19087538.242243454
iteration 10 WCSS =  18596517.368090734
iteration 11 WCSS =  18083614.717021115
iteration 12 WCSS =  17596459.88655153
iteration 13 WCSS =  17157083.371140227
iteration 14 WCSS =  16711904.483522398
iteration 15 WCSS =  16296609.056771155
iteration 16 WCSS =  15964166.881750261
iteration 17 WCSS =  15709637.325473953
iteration 18 WCSS =  15471680.80901358
iteration 19 WCSS =  15204757.67213304
iteration 20 WCSS =  14880927.562267441
iteration 21 WCSS =  14500658.953930428
iteration 22 WCSS =  14194685.86238497
iteration 23 WCSS =  13976880.939191196
```

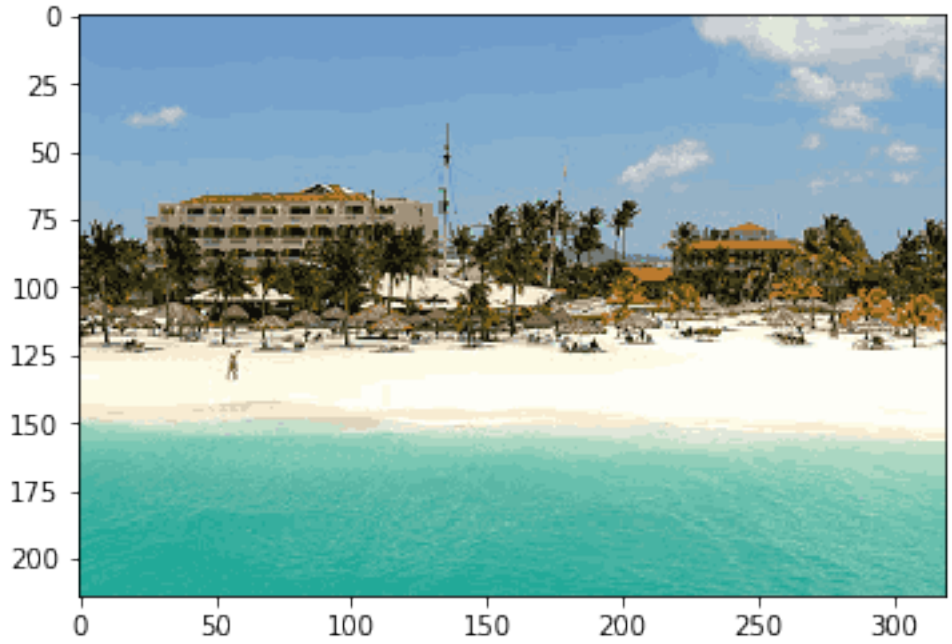
```
iteration 24 WCSS =  13838267.36840708
iteration 25 WCSS =  13738525.530358797
iteration 26 WCSS =  13667216.899612563
iteration 27 WCSS =  13609941.36692111
iteration 28 WCSS =  13557017.495982805
iteration 29 WCSS =  13508680.635023192
iteration 30 WCSS =  13453754.620556723
iteration 31 WCSS =  13391642.920642532
iteration 32 WCSS =  13326003.705468126
iteration 33 WCSS =  13258539.167323949
iteration 34 WCSS =  13181681.684857016
iteration 35 WCSS =  13121533.884225918
iteration 36 WCSS =  13072574.633198025
iteration 37 WCSS =  13024967.759802537
iteration 38 WCSS =  12979457.397783194
iteration 39 WCSS =  12938028.339539427
iteration 40 WCSS =  12893932.894749936
iteration 41 WCSS =  12853989.8608272
iteration 42 WCSS =  12816882.375179872
iteration 43 WCSS =  12783196.339708138
iteration 44 WCSS =  12752950.188900907
iteration 45 WCSS =  12728497.258673282
iteration 46 WCSS =  12708456.502574416
iteration 47 WCSS =  12693461.341536349
iteration 48 WCSS =  12679884.31734222
iteration 49 WCSS =  12666346.084923895
iteration 50 WCSS =  12654851.480446061
iteration 51 WCSS =  12643597.608039401
iteration 52 WCSS =  12632437.932018476
iteration 53 WCSS =  12621036.359780535
iteration 54 WCSS =  12610957.200396188
iteration 55 WCSS =  12602623.40389106
iteration 56 WCSS =  12594842.284573963
iteration 57 WCSS =  12587285.053839942
iteration 58 WCSS =  12580624.025513945
iteration 59 WCSS =  12573782.713509941
iteration 60 WCSS =  12569531.367298346
iteration 61 WCSS =  12564912.087189432
iteration 62 WCSS =  12561972.234068686
iteration 63 WCSS =  12559028.771523768
iteration 64 WCSS =  12556365.315948239
iteration 65 WCSS =  12553658.957864676
iteration 66 WCSS =  12551113.593543626
iteration 67 WCSS =  12547916.846415263
iteration 68 WCSS =  12544753.563985482
iteration 69 WCSS =  12540657.294862336
iteration 70 WCSS =  12536371.203422928
iteration 71 WCSS =  12530310.772718543
```

```
iteration 72 WCSS =  12521658.383171786
iteration 73 WCSS =  12510027.980022606
iteration 74 WCSS =  12494904.039293844
iteration 75 WCSS =  12476450.053978875
iteration 76 WCSS =  12455106.31783842
iteration 77 WCSS =  12432651.74851378
iteration 78 WCSS =  12413832.968960686
iteration 79 WCSS =  12395922.285882289
iteration 80 WCSS =  12376816.994168688
iteration 81 WCSS =  12358248.545607967
iteration 82 WCSS =  12340274.380309071
iteration 83 WCSS =  12315343.834254948
iteration 84 WCSS =  12292036.912821546
iteration 85 WCSS =  12269577.209209476
iteration 86 WCSS =  12249175.374485949
iteration 87 WCSS =  12228867.636745503
iteration 88 WCSS =  12211966.266162788
iteration 89 WCSS =  12195395.544180887
iteration 90 WCSS =  12182709.356720056
iteration 91 WCSS =  12173153.922856841
iteration 92 WCSS =  12164247.407576885
iteration 93 WCSS =  12154601.109877376
iteration 94 WCSS =  12146379.77442271
iteration 95 WCSS =  12139747.243896605
iteration 96 WCSS =  12133019.465266364
iteration 97 WCSS =  12127325.836457692
iteration 98 WCSS =  12122217.621295631
iteration 99 WCSS =  12118095.249393994
iteration 100 WCSS =  12115061.937037438
iteration 101 WCSS =  12112725.576144315
iteration 102 WCSS =  12110957.989098338
iteration 103 WCSS =  12109251.112020874
iteration 104 WCSS =  12107594.093575923
iteration 105 WCSS =  12105480.474864507
iteration 106 WCSS =  12103114.791187843
iteration 107 WCSS =  12101205.29481378
iteration 108 WCSS =  12098852.070808975
iteration 109 WCSS =  12096795.020683402
iteration 110 WCSS =  12094439.687005566
iteration 111 WCSS =  12091584.712140342
iteration 112 WCSS =  12088315.001579128
iteration 113 WCSS =  12085773.359141104
iteration 114 WCSS =  12083413.599208545
iteration 115 WCSS =  12080896.27714626
iteration 116 WCSS =  12078720.31485486
iteration 117 WCSS =  12077292.383412044
iteration 118 WCSS =  12076233.799854983
iteration 119 WCSS =  12074633.330525931
```

```
iteration 120 WCSS =  12072864.15654355
iteration 121 WCSS =  12071634.495801235
iteration 122 WCSS =  12070079.262982232
iteration 123 WCSS =  12067935.096918315
iteration 124 WCSS =  12065411.869487232
iteration 125 WCSS =  12062118.600492494
iteration 126 WCSS =  12058068.066725899
iteration 127 WCSS =  12053955.1898843
iteration 128 WCSS =  12050161.897327507
iteration 129 WCSS =  12045587.005010098
iteration 130 WCSS =  12040835.827572871
iteration 131 WCSS =  12036011.03797824
iteration 132 WCSS =  12030041.637581395
iteration 133 WCSS =  12024608.989578852
iteration 134 WCSS =  12018160.343562547
iteration 135 WCSS =  12011181.83984205
iteration 136 WCSS =  12004893.777427336
iteration 137 WCSS =  11999183.757419096
iteration 138 WCSS =  11992626.871406361
iteration 139 WCSS =  11986127.531601181
iteration 140 WCSS =  11980651.05377796
iteration 141 WCSS =  11976176.189491648
iteration 142 WCSS =  11972315.458984416
iteration 143 WCSS =  11969092.549933357
iteration 144 WCSS =  11966930.44005058
iteration 145 WCSS =  11965469.449237531
iteration 146 WCSS =  11964295.841543075
iteration 147 WCSS =  11963100.489264412
iteration 148 WCSS =  11962625.516585309
iteration 149 WCSS =  11962181.211487884
iteration 150 WCSS =  11960957.652884718
iteration 151 WCSS =  11960122.652047219
iteration 152 WCSS =  11959487.474610144
iteration 153 WCSS =  11958502.777246289
iteration 154 WCSS =  11957772.83604165
iteration 155 WCSS =  11957111.627938602
iteration 156 WCSS =  11956766.545048788
iteration 157 WCSS =  11956194.64685188
iteration 158 WCSS =  11955522.094592327
iteration 159 WCSS =  11954949.914927276
iteration 160 WCSS =  11954595.92319678
iteration 161 WCSS =  11954322.64258102
iteration 162 WCSS =  11954027.633079138
iteration 163 WCSS =  11953756.897055842
iteration 164 WCSS =  11953561.359836362
iteration 165 WCSS =  11953394.150561273
iteration 166 WCSS =  11953143.744293556
iteration 167 WCSS =  11952872.0460485
```

```
iteration 168 WCSS =  11952726.840225533
iteration 169 WCSS =  11952607.951913094
iteration 170 WCSS =  11952485.861721046
iteration 171 WCSS =  11952399.74689186
iteration 172 WCSS =  11952253.03103106
iteration 173 WCSS =  11952022.830360595
iteration 174 WCSS =  11951854.329699213
iteration 175 WCSS =  11951421.764006669
iteration 176 WCSS =  11950859.074740596
iteration 177 WCSS =  11950242.422732463
iteration 178 WCSS =  11949566.615759084
iteration 179 WCSS =  11949180.753478503
iteration 180 WCSS =  11949017.025349438
iteration 181 WCSS =  11948675.964173885
iteration 182 WCSS =  11948249.065074338
iteration 183 WCSS =  11947859.993417114
iteration 184 WCSS =  11947490.178907802
iteration 185 WCSS =  11947088.728614189
iteration 186 WCSS =  11946867.108135125
iteration 187 WCSS =  11946631.074624576
iteration 188 WCSS =  11946334.08826181
iteration 189 WCSS =  11946171.448187334
iteration 190 WCSS =  11946089.118143868
iteration 191 WCSS =  11946064.301593367
iteration 192 WCSS =  11946045.449806698
iteration 193 WCSS =  11946000.293923449
iteration 194 WCSS =  11945959.450258758
iteration 195 WCSS =  11945928.799474988
iteration 196 WCSS =  11945911.698249673
iteration 197 WCSS =  11945905.71659477
iteration 198 WCSS =  11945899.023637032
iteration 199 WCSS =  11945897.552198278
Time taken = 391.71 seconds
Image with k = 32 clusters…
The labels are: [11 11 11 … 29 29 29]
The cluster centers are {0: array([117.14167813, 123.39477304, 121.93122421]),
1: array([14.69779853, 13.80386925,  8.8305537 ]), 2: array([248.48988064,
236.99844318, 210.39543332]), 3: array([170.81076389, 215.015625  ,
194.26822917]), 4: array([253.40602786, 251.17401194, 230.14444129]), 5:
array([45.48518851, 43.66337522, 19.502693  ]), 6: array([134.17755573,
113.59569562,  84.44504228]), 7: array([223.9772118 , 212.23458445,
188.26541555]), 8: array([138.56654306, 177.5039143 , 205.5986815 ]), 9:
array([149.07262731, 178.81105324, 198.67332176]), 10: array([127.92953822,
171.22651274, 206.40067675]), 11: array([111.18968583, 156.69176052,
202.94694724]), 12: array([224.78610603, 231.79616088, 218.34277879]), 13:
array([167.67966102, 185.94237288, 200.81949153]), 14: array([187.21122112,
122.51320132,  46.10066007]), 15: array([144.50282087, 206.01622003,
185.27433004]), 16: array([188.38696109, 196.21556257, 202.21451104]), 17:
```

```
array([121.65080875,  94.39105614,  25.77069458]), 18: array([ 54.11405836,
174.60515347, 158.47063282]), 19: array([204.72897196, 209.70560748,
214.70794393]), 20: array([ 71.50782313, 179.78707483, 163.26054422]), 21:
array([197.35307517, 224.03075171, 202.8166287 ]), 22: array([94.89716546,
88.19644034, 66.18193804]), 23: array([121.8874669 , 199.42365402,
179.94042365]), 24: array([ 34.42066077, 169.17543043, 152.22056771]), 25:
array([160.83688833, 145.4523212 , 122.42346299]), 26: array([73.32458126,
67.9932096 , 30.1611589 ]), 27: array([119.11422992, 163.84159141,
204.86331462]), 28: array([103.72065889, 191.21551132, 173.6273164 ]), 29:
array([ 90.03983516, 182.20192308, 165.75549451]), 30: array([254.13774717,
254.47145079, 248.99022439]), 31: array([190.88261351, 174.28460687,
150.12403101])}
```



[ ]:

HW1 Q2 K-medoids Arjun Singh

May 24, 2020

[113]:
```
import numpy as np
import time
```

[114]:
```
def init_medoids(X, k):
#   Randomly select k points from the image as the starting centers
    centers = np.random.choice(len(X),k,replace=False)
#     print(np.repeat(centers,k))
#     temp = np.repeat(centers,k)
#     return X[temp,:]
    return X[centers,:]
```

[115]:
```
# Compute  distance. I've included the metric p in the input to allow various␣
 ↪norms to compute the
# distance (rather than just Euclidean)
def compute_distance(X, centers, p):
    m = len(X)

    center_shape = centers.shape
    if len(center_shape)==1:
        centers = centers.reshape((1,len(centers)))

    s=len(centers)

    S = np.empty((m,s))
    for i in range(m):
        S[i,:] = np.linalg.norm(X[i,:]-centers,ord=p,axis=1)**p
    return S
```

[116]:
```
# Another approach to calculating distance between points in cluster. Avoids␣
 ↪the loop in the previous distance
# calculation approach.
def distance_calc (cluster_points,center,p,k):
    dist = []
    label = []
    min_dist = []

    main_array = list(range(k))
```

```
    pixel_temp = np.tile(center,(len(cluster_points),1))

    main_array= np.linalg.norm(cluster_points - pixel_temp, ord=p,axis=1)
    min_dist = main_array

    return min_dist
```

[117]:
```
def assign_cluster_labels(S):
    return np.argmin(S,axis=1)
```

[118]:
```
# Updates the cluster centers by finding the point with the least distance to␣
 ↪all other points.
def update_centers(X, centers, p,k):

    S = compute_distance(X, centers,p)
    labels = assign_cluster_labels(S)

    curr_centers = centers
#   Iterate over all clusters
    for i in set(labels):
        cluster_points = X[labels==i]
        cluster_points = np.unique(cluster_points,axis=0)
        avg_distance = np.sum(distance_calc(cluster_points, centers[i],p,k))
#       Iterate over all points in each cluster
        for points in cluster_points:
            new_distance = np.sum(distance_calc(cluster_points,points,p,k))
#           If distance from current point to all other points in the cluster␣
 ↪is lower than previous minimum
#           then update cluster center to the new point

            if new_distance < avg_distance:
                avg_distance = new_distance
                curr_centers[i] = points

    return curr_centers
```

[119]:
```
# Calculate the within clusters sum of squares
def WCSS(S):
    return np.sum(np.amin(S,axis=1))
```

[121]:
```
def kmedoids(X, k,p,starting_centers=None,max_steps=np.inf):
    start = time.time()
#   Begin by initializing starting points
    if starting_centers is None:
        centers = init_medoids(X, k)
    else:
        centers = starting_centers
```

```
    converged = False
    labels = np.zeros(len(X))
    i = 1
    wc= 99999999999
#   Begin loop for algorithm
    while (not converged) and (i <= max_steps):
        oldwcss = wc
        old_centers = centers

        S = compute_distance(X, old_centers,p)

        labels = assign_cluster_labels(S)

        centers = update_centers(X, centers,p,k)

        wc = WCSS(S)
#       If current step's WCSS is less than a 5% imporvement over previous␣
 ↪step, terminate
        if wc > 0.95*oldwcss:
            converged = True
        else:
            converged = False
        print ("iteration", i, "WCSS = ", WCSS (S))
        i += 1

    stop = time.time()
    print("Time taken = {:.2f} seconds".format(stop-start))
    return labels
```

[62]:
```
from matplotlib.pyplot import imshow
%matplotlib inline
def display_image(arr):
    """
    display the image
    input : 3 dimensional array
    """
    arr = arr.astype(dtype='uint8')
    img = Image.fromarray(arr, 'RGB')
    imshow(np.asarray(img))
```

[63]:
```
from PIL import Image
def read_img(path):
    """
    Read image and store it as an array, given the image path.
    Returns the 3 dimensional image array.
    """
```

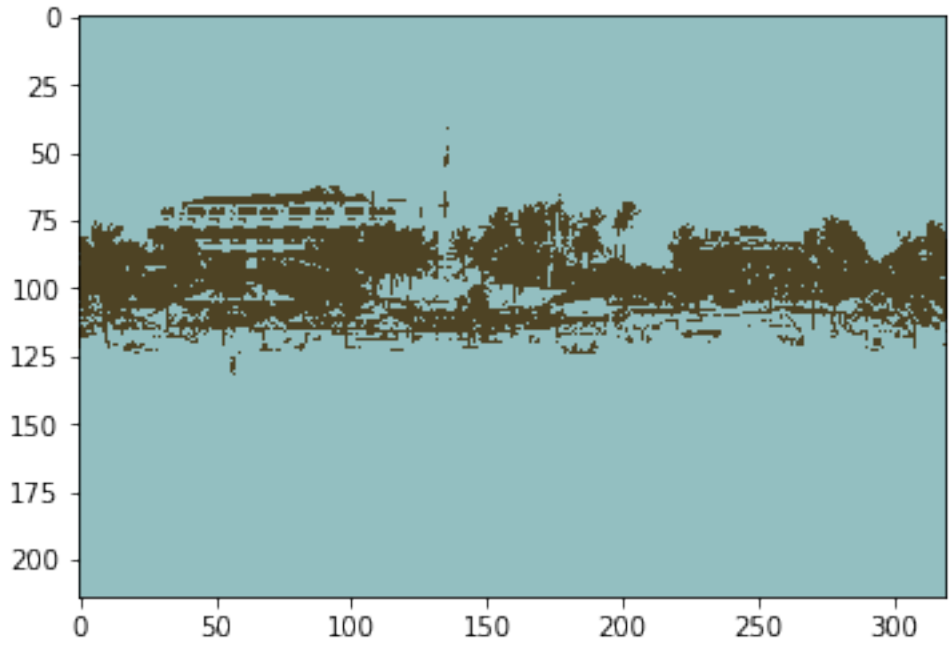
```
    img = Image.open(path)
    img_arr = np.array(img, dtype='int32')
#     img_arr = np.array(img.getdata())

    img.close()
    return img_arr
```

[64]:
```
def runKMedoidsAlgorithm(pixels,k):
    print("Running k-medoids algorithm...")
#   Read in image
    image = read_img(pixels)
    r, c, l = image.shape
#   Flatten image
    img_reshaped = np.reshape(image, (r*c, l), order="C")
#   Choose norm criteria
    p=2
#   If the value k is too high, reduce it
    if k>= (len(img_reshaped)/3):
        k = (len(img_reshaped)/3)

#   Run algorithm
    labels = kmedoids(img_reshaped, k,p, starting_centers=None)
    ind = np.column_stack((img_reshaped, labels))
    centers = {}
    for i in set(labels):
        c = ind[ind[:,3] == i].mean(axis=0)
        centers[i] = c[:3]

    img_clustered = np.array([centers[i] for i in labels])
    r, c, l = image.shape
    img_disp = np.reshape(img_clustered, (r, c, l), order="C")

    print('Image with k = ' + str(k) + " clusters...")
    display_image(img_disp)

    print("The labels are: {}".format(labels.T))
    print("The cluster centers are {}".format(centers))

    return labels,centers
```

[104]:
```
lab, cen = runKMedoidsAlgorithm("data/beach.bmp",2)
```

```
Running k-medoids algorithm…
iteration 1 WCSS =  1294971087.0
iteration 2 WCSS =  437519881.0
iteration 3 WCSS =  412498211.0
iteration 4 WCSS =  412002293.0
```

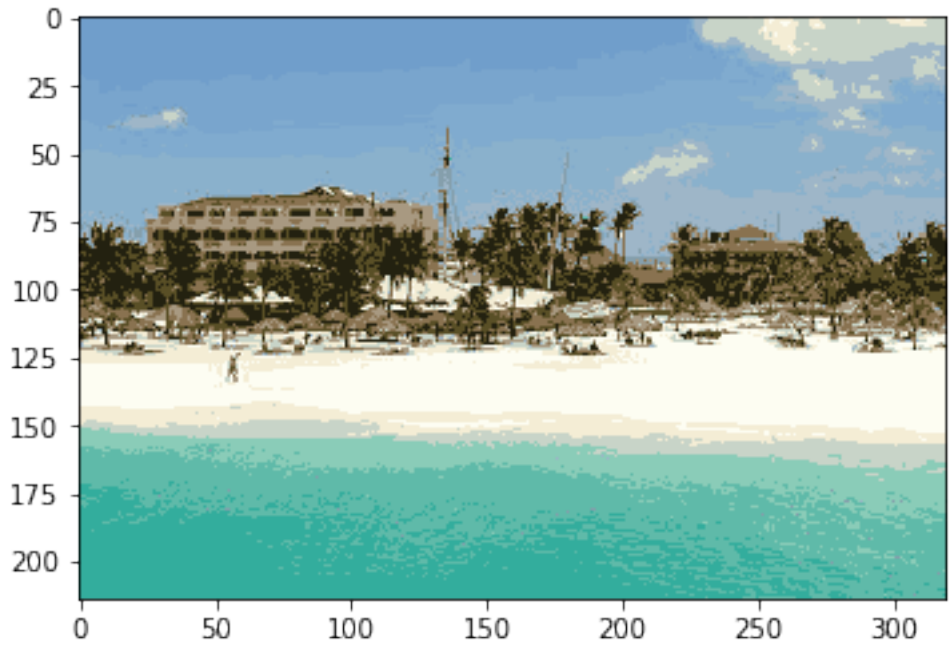
```
Time taken = 85.63 seconds
Image with k = 2 clusters…
The labels are: [0 0 0 … 0 0 0]
The cluster centers are {0: array([147.16211718, 191.53710217, 193.77184524]),
1: array([78.19784208, 67.69475233, 36.66287396])}
```



[105]:
```
lab, cen = runKMedoidsAlgorithm("data/beach.bmp",16)
```

```
Running k-medoids algorithm…
iteration 1 WCSS =  39547446.0
iteration 2 WCSS =  29251741.0
iteration 3 WCSS =  28094872.0
Time taken = 25.94 seconds
Image with k = 16 clusters…
The labels are: [ 9  9  9 … 14 14 14]
The cluster centers are {0: array([138.93415033, 177.11029412, 204.60669935]),
1: array([128.04694206, 171.7360515 , 207.2194206 ]), 2: array([158.61106899,
182.99620925, 200.34571645]), 3: array([133.44534413, 159.74898785,
172.53846154]), 4: array([138.28383562, 204.64356164, 184.32027397]), 5:
array([168.84196185, 153.09264305, 129.19663942]), 6: array([244.19209848,
237.29716576, 213.98854853]), 7: array([121.14083333, 165.68666667,
205.32805556]), 8: array([ 51.49277347, 173.9493382 , 157.42933212]), 9:
array([112.45182434, 158.05724274, 203.68756828]), 10: array([200.41001221,
212.13626374, 200.37338217]), 11: array([89.62761613, 79.46579888,
38.39841756]), 12: array([253.87523681, 253.52002706, 242.33667118]), 13:
array([35.378612  , 33.70560632, 16.00740924]), 14: array([ 95.11442588,
```

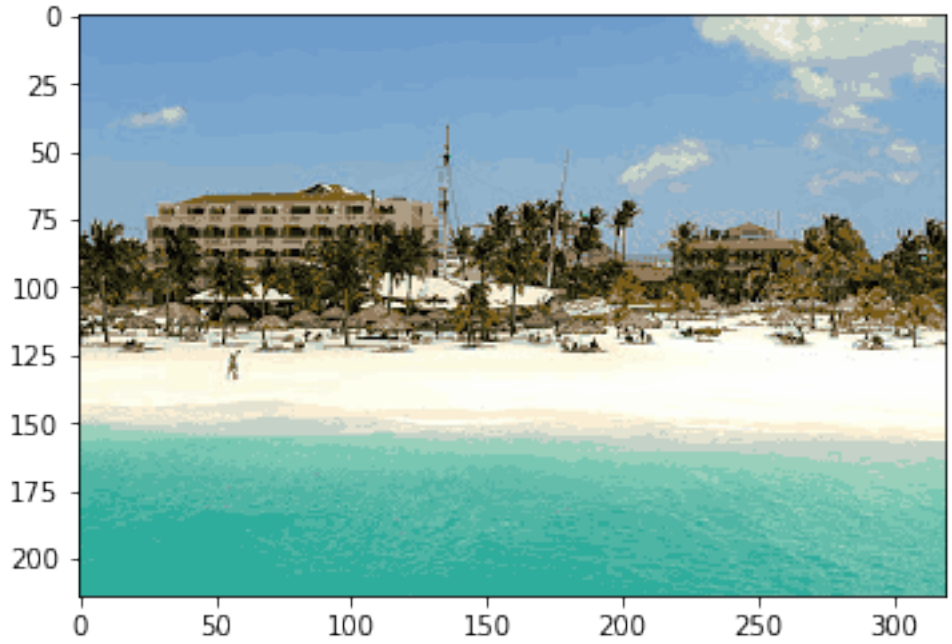
```
186.68443384, 169.57703527]), 15: array([137.79430789, 114.00258732,
80.36578266])}
```



[106]:
```
lab, cen = runKMedoidsAlgorithm("data/beach.bmp",32)
```

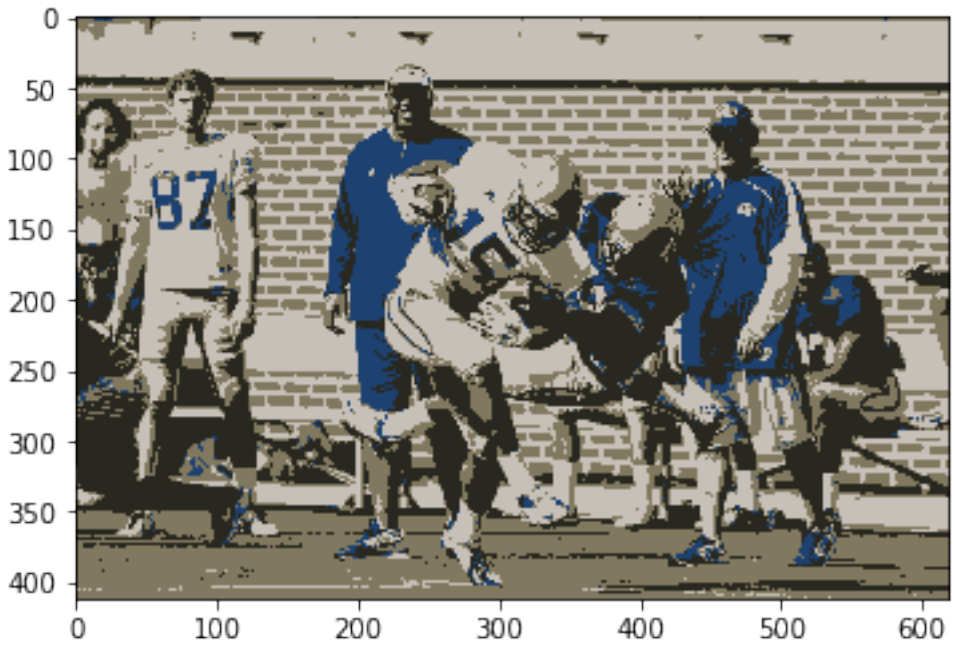
```
Running k-medoids algorithm…
iteration 1 WCSS =  40365040.0
iteration 2 WCSS =  25836544.0
iteration 3 WCSS =  21509027.0
iteration 4 WCSS =  18861152.0
iteration 5 WCSS =  17949460.0
Time taken = 37.14 seconds
Image with k = 32 clusters…
The labels are: [9 9 9 … 6 6 6]
The cluster centers are {0: array([102.61856254,  93.72778721,  72.14714205]),
1: array([123.60374314, 167.72668603, 206.10519522]), 2: array([133.08716707,
154.8716707 , 166.63680387]), 3: array([253.91415577, 254.14383172,
238.14837976]), 4: array([169.11556728, 185.73192612, 198.78416887]), 5:
array([253.593361  , 250.41138115, 226.5429757 ]), 6: array([100.45764167,
188.36119061, 171.1044648 ]), 7: array([141.13195876, 175.62852234, 199.2628866
]), 8: array([3.29399586, 2.33747412, 4.94202899]), 9: array([110.36245016,
156.20188474, 203.12794491]), 10: array([228.62790698, 227.20237506,
209.07174666]), 11: array([120.51625306, 199.07375044, 179.69136666]), 12:
array([254.45564893, 254.60037348, 252.49579832]), 13: array([196.77697161,
210.73911672, 200.34794953]), 14: array([146.29267613, 180.89334447,
205.13951546]), 15: array([70.88827586, 66.62482759, 34.08367816]), 16:
```

```
array([48.80955088, 46.22285389, 18.72370665]), 17: array([254.95708155,
254.79828326, 254.98998569]), 18: array([152.08618899, 208.5768432 ,
187.72689512]), 19: array([131.00260473, 173.88083351, 207.39830692]), 20:
array([240.41935484, 241.73387097, 241.28225806]), 21: array([253.9470437 ,
254.50796915, 247.85141388]), 22: array([121.25098296,  93.51572739,
25.62057667]), 23: array([147.44496991, 124.08340499,  93.03568358]), 24:
array([251.81137725, 249.81137725, 232.24700599]), 25: array([117.06461445,
162.18957635, 204.65650716]), 26: array([250.5407226 , 239.955297  ,
213.11757502]), 27: array([ 78.53684448, 180.65257685, 164.23915009]), 28:
array([27.68958743, 27.79174853, 14.38212181]), 29: array([14.37115839,
12.356974  ,  7.64539007]), 30: array([173.98724348, 157.92956184, 133.5890183
]), 31: array([ 46.63374007, 172.5704048 , 156.07883014])}
```



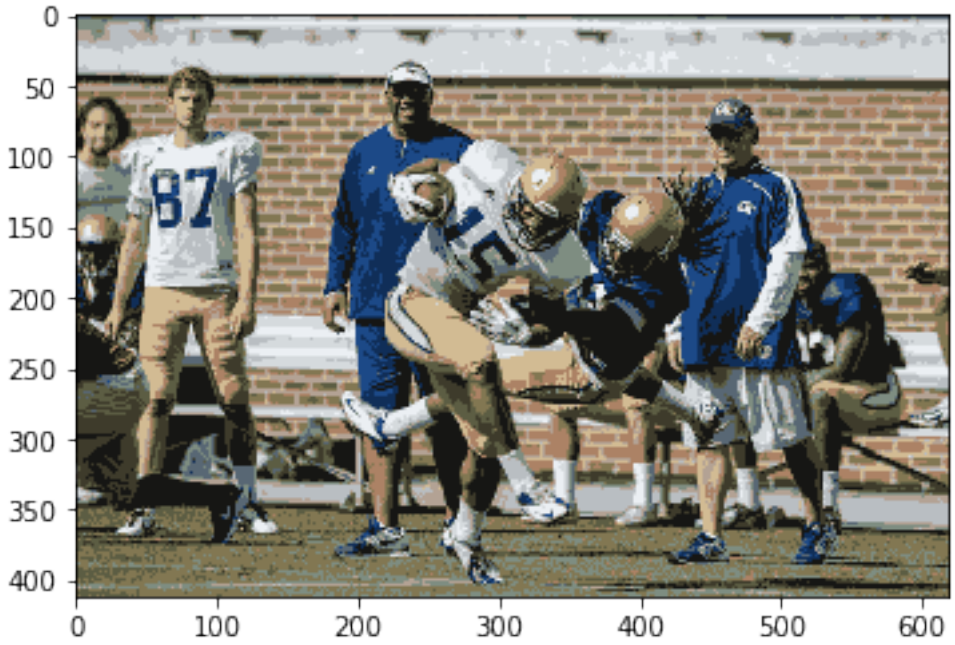
[107]:
```
lab, cen = runKMedoidsAlgorithm("data/football.bmp",4)
```

```
Running k-medoids algorithm…
iteration 1 WCSS =  795086474.0
iteration 2 WCSS =  653969572.0
iteration 3 WCSS =  627066380.0
Time taken = 380.82 seconds
Image with k = 4 clusters…
The labels are: [0 0 0 … 1 1 1]
The cluster centers are {0: array([ 29.59608288,  65.62690888, 112.67482259]),
1: array([129.45329423, 121.12920635,  95.33023646]), 2: array([198.27617676,
192.57896966, 184.46228852]), 3: array([41.51326068, 40.64565853, 32.77022857])}
```



[108]:
```
lab, cen = runKMedoidsAlgorithm("data/football.bmp",14)
```

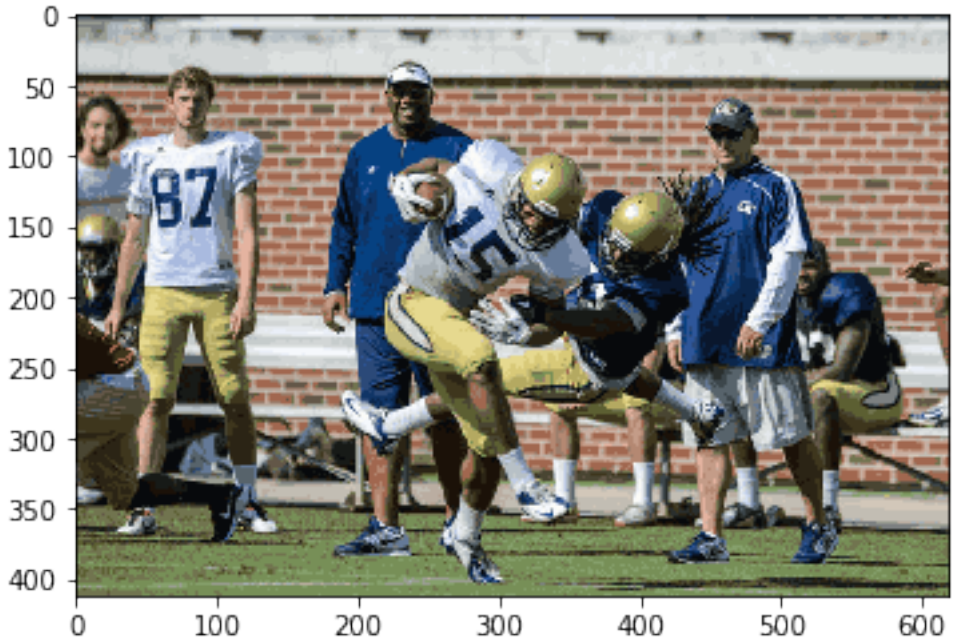
```
Running k-medoids algorithm…
iteration 1 WCSS =  300888819.0
iteration 2 WCSS =  177207394.0
iteration 3 WCSS =  171816200.0
Time taken = 153.89 seconds
Image with k = 14 clusters…
The labels are: [0 0 0 … 8 8 8]
The cluster centers are {0: array([92.10732696, 93.67772951, 67.18986389]), 1:
array([172.23467107, 130.04667873, 101.3762508 ]), 2: array([ 88.55267126,
103.30450237, 116.30116329]), 3: array([210.25700383, 187.12112946,
147.51408139]), 4: array([213.62941392, 218.83488779, 221.56828086]), 5:
array([185.54045959, 190.50844043, 194.81077107]), 6: array([179.07417878,
162.7141947 , 148.90364324]), 7: array([235.88742656, 239.81786134,
243.79059929]), 8: array([130.36939915, 119.26123379,  82.05742072]), 9:
array([22.00743449, 22.11973768, 20.50276527]), 10: array([16.08882979,
46.85638298, 88.30159574]), 11: array([ 28.08408743,  73.822776  ,
135.71098427]), 12: array([62.60444485, 59.38514462, 43.01787897]), 13:
array([126.16531975, 141.5979152 , 129.76446475])}
```



[109]:
```
lab, cen = runKMedoidsAlgorithm("data/football.bmp",50)
```

```
Running k-medoids algorithm…
iteration 1 WCSS =  105560989.0
iteration 2 WCSS =  66334801.0
iteration 3 WCSS =  60459670.0
iteration 4 WCSS =  57889609.0
Time taken = 130.47 seconds
Image with k = 50 clusters…
The labels are: [ 6 35 35 … 43 43 43]
The cluster centers are {0: array([132.31554054, 148.33006757, 175.23108108]),
1: array([180.16598192, 169.34552177,  97.80073952]), 2: array([210.48488603,
214.78580278, 214.3327552 ]), 3: array([169.98135755, 186.3001912 , 223.1042065
]), 4: array([167.67774011, 176.62079096, 182.03954802]), 5:
array([201.16790831, 175.97191977, 161.0913085 ]), 6: array([ 87.305041  ,
99.62131795, 107.39386578]), 7: array([230.83074451, 217.69523299,
143.39314408]), 8: array([252.68463612, 252.23315364, 248.58490566]), 9:
array([42.55396766, 47.48401655, 46.44697255]), 10: array([183.07935504,
192.170724  , 198.70882074]), 11: array([23.73424271, 26.4474757 ,
23.37284415]), 12: array([124.17273559, 128.51125343, 129.92369625]), 13:
array([ 16.24119672,  53.33545142, 102.35901509]), 14: array([181.97880006,
160.04752667, 145.29333518]), 15: array([134.33531915, 160.0470922 ,
108.10695035]), 16: array([229.22205551, 233.73543386, 237.02175544]), 17:
array([197.50097213, 187.7433571 , 119.13480233]), 18: array([161.55821213,
103.100352  ,  79.94979079]), 19: array([ 7.33829224,  9.51609945,
12.02761361]), 20: array([154.90515873, 157.31626984, 146.61646825]), 21:
```

array([245.70860566, 248.66557734, 252.03213508]), 22: array([210.85182768,
222.4575718 , 243.97845953]), 23: array([186.30210773, 173.65858147,
161.66711275]), 24: array([216.01164144, 201.99592549, 120.56111758]), 25:
array([43.48606688, 73.02707006, 120.28264331]), 26: array([46.40448962,
86.74544684, 153.87462939]), 27: array([100.43980222, 121.92830655,
151.32682324]), 28: array([62.68467078, 54.32896091, 29.38271605]), 29:
array([73.79702048, 69.19180633, 50.62876547]), 30: array([180.21917627,
136.01946762, 114.9188187]), 31: array([194.78248281, 189.1514658 ,
179.87513572]), 32: array([192.21796835, 205.91985707, 231.78815722]), 33:
array([155.58940043, 126.64864383, 108.10331906]), 34: array([11.81457663,
38.13210075, 75.8204716]), 35: array([60.21021963, 70.28955625, 75.22052891]),
36: array([10.24493164, 23.17586044, 45.79561528]), 37: array([198.62847409,
204.15545992, 204.56051641]), 38: array([221.56443935, 225.27626939,
224.60331453]), 39: array([210.67559944, 199.67207334, 185.29830748]), 40:
array([6.25477154, 65.43175246, 129.51908618]), 41: array([92.08281999,
95.75952544, 50.52589551]), 42: array([234.69634703, 239.95091324,
248.67522831]), 43: array([119.98592383, 143.43083602, 87.98191146]), 44:
array([43.49216783, 36.86587413, 21.60965035]), 45: array([228.13007457, 179.
, 160.24440762]), 46: array([196.59147916, 156.12098857, 136.65805976]), 47:
array([113.12531306, 116.91878354, 79.859839]), 48: array([99.49529988,
87.2148815 , 72.15305177]), 49: array([27.02577622, 17.34094903, 8.72075766])}



HW1 Q3

May 24, 2020

[25]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
from scipy import sparse
from matplotlib import pyplot as plt
import networkx as nx
from sklearn.cluster import KMeans
```

[26]:
```
# Reading in the data
nodes = pd.read_csv('data/nodes.txt',sep="\t", header=None)
edges = pd.read_csv('data/edges.txt',sep="\t",header=None)
```

[27]:
```
# Data exploration and cleaning
print(nodes.isnull().values.any())
print(nodes.isnull().sum())
```

```
True
0    0
1    0
2    2
3    2
dtype: int64
```

[28]:
```
missingRows = nodes[nodes[2].isnull()]
# Nodes 56 and 111 are missing values
print(missingRows)
# We can remove these 2 points from our dataset
print(len(nodes))
cleanNodes = nodes.dropna()
print(len(cleanNodes))
```

```
       0                                          1   2    3
55    56  atrios.blogspot.com/\t0\tLabeledManually" NaN  NaN
110  111   brunon.blogspot.com\t0\tLabeledManually" NaN  NaN
1490
1488
```

[29]:
```
uniqueNodes = cleanNodes[0].unique()
```

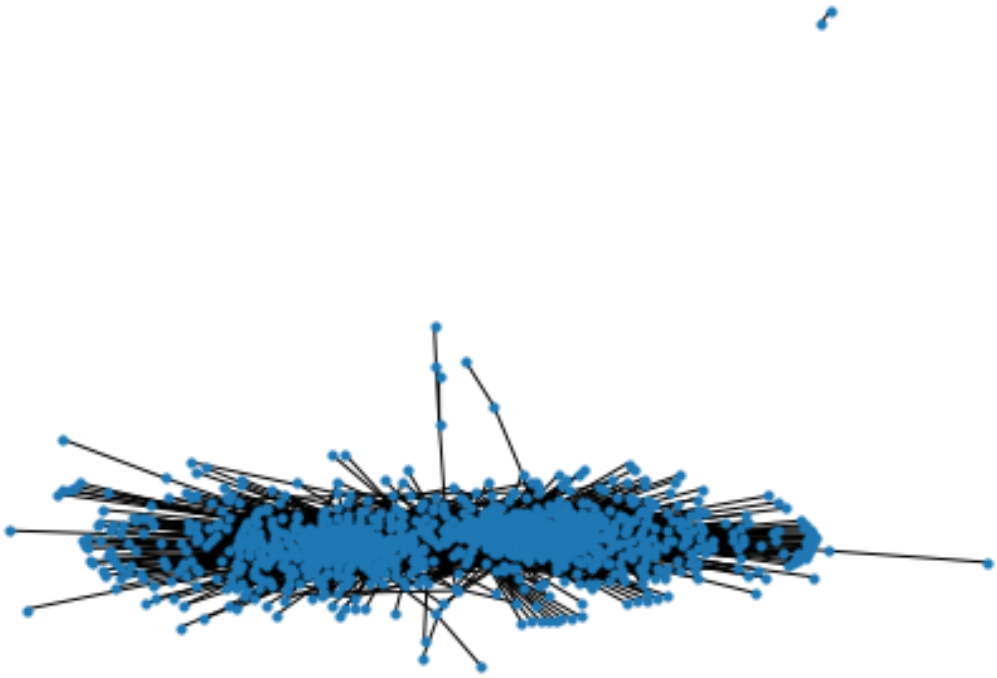
[30]:
```
cleanEdges = edges
```

[31]:
```
# Removing the missing nodes from the edges list as well
cleanEdges = cleanEdges[(cleanEdges[0].isin(uniqueNodes)) & (cleanEdges[1].
 ↪isin(uniqueNodes))]
```

[32]:
```
len(cleanEdges)
```

[32]: 19002

[33]:
```
# Graphing the data
g_data=nx.from_pandas_edgelist(cleanEdges, 0, 1)
nx.draw(g_data, node_size = 10)
plt.show()
```

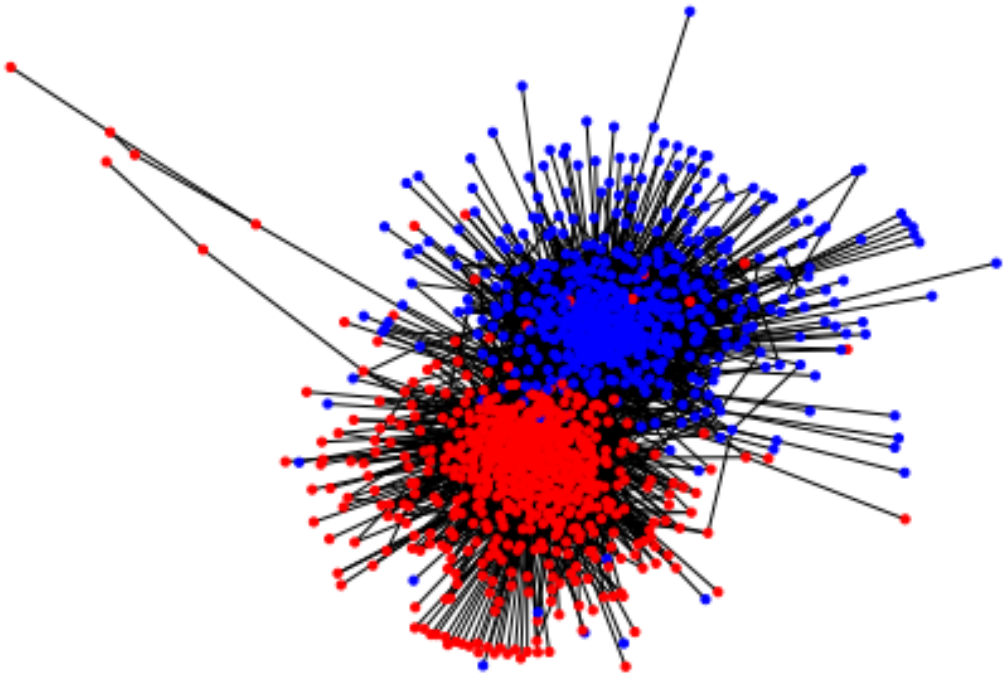


[34]:
```
# Finding the isolated nodes
components = list(nx.connected_components(g_data))
components.sort(key=len, reverse=True)
largest = components.pop(0)
num_isolated = g_data.order() - len(largest)
print("Number of isolated nodes = " + str(num_isolated) + " at " +␣
 ↪str(components))
```

```
Number of isolated nodes = 2 at [{666, 182}]
```

[35]:
```
# Removing the isloated nodes from the edgelist on the nodelist
cleanEdges2 = cleanEdges[~(cleanEdges[0].isin([182,666])) & ~(cleanEdges[1].
 ↪isin([182,666]))]
cleanNodes2 = cleanNodes[~cleanNodes[0].isin([182,166])]
```

[36]:
```
# Plotting a graph to show the political orientation of the blogs
g_data=nx.from_pandas_edgelist(cleanEdges2, 0, 1)
# Adding color for each orientation
colorList = []
for each in g_data:
    if int(nodes[nodes[0]==each][2]) == 0:
        colorList.append('blue')
    else:
        colorList.append('red')
nx.draw(g_data, node_size = 10,node_color=colorList)
plt.show()
```



[49]:
```
# Beginning the spectral clustering algorithm
k = 2
n = max(cleanEdges2[0])
a = np.array(cleanEdges2)
```

[50]:
```
i = a[:, 0]-1
j = a[:, 1]-1
v = np.ones((a.shape[0], 1)).flatten()

A = sparse.coo_matrix((v, (i, j)), shape=(n, n))
A = (A + np.transpose(A))/2
```

[51]:
```
# M = A[A.getnnz(1)>0][:,A.getnnz(0)>0]
M = A
```

[52]:
```
D = np.diag(1/np.sqrt(np.sum(M, axis=1)).A1)
# L = D @ M @ D
# D = np.diag(np.sum(M, axis=1))
L = D - M
L = np.nan_to_num(L)
print(D)
v, x = np.linalg.eig(L)
x = x[:, 0:k].real
copyx = x
# x = np.squeeze(np.asarray(x))
# x = x/np.repeat(np.sqrt(np.sum(x*x, axis=1).reshape(-1, 1)), k, axis=1)

# scatter
# plt.scatter(x[:, 0], x[:, 1])
# plt.show()
```

```
/Users/ajspsp/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:
RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

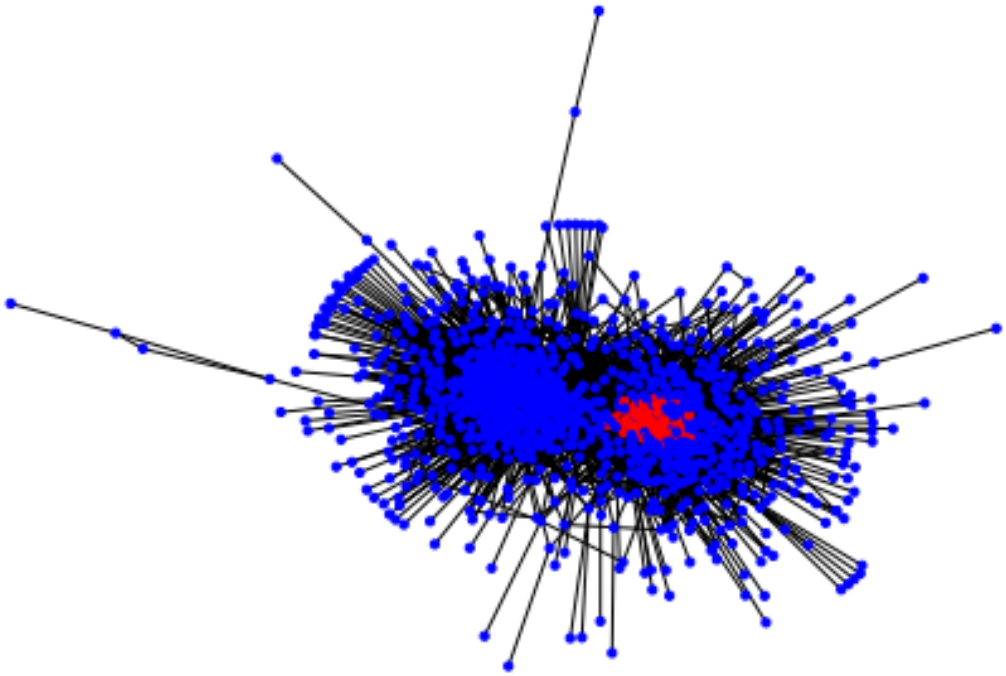
```
[[0.27216553 0.         0.         … 0.         0.         0.        ]
 [0.         0.20412415 0.         … 0.         0.         0.        ]
 [0.         0.                inf … 0.         0.         0.        ]
 …
 [0.         0.         0.         … 1.41421356 0.         0.        ]
 [0.         0.         0.         … 0.         0.30151134 0.        ]
 [0.         0.         0.         … 0.         0.         1.41421356]]
```

[53]:
```
kmeans = KMeans(n_clusters=2).fit(copyx)
idx = kmeans.labels_

df = nodes.copy()
```

[54]:
```
df['Predicted'] = idx
predictedNodes = df.dropna()
```

[55]:
```
# Plotting a graph to show the political orientation of the blogs from the␣
 ↪predicted values
predictedData=nx.from_pandas_edgelist(cleanEdges2, 0, 1)
# Adding color for each orientation
predictedcolorList = []

for each in predictedData:
    if (df[df[0]==each]['Predicted'].values[0]) == 0:
        predictedcolorList.append('blue')
    else:
        predictedcolorList.append('red')
nx.draw(predictedData, node_size = 10,node_color=predictedcolorList)
plt.show()
```

```
/Users/ajspsp/opt/anaconda3/lib/python3.7/site-
packages/networkx/drawing/nx_pylab.py:579: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use np.iterable instead.
  if not cb.iterable(width):
```



[56]:
```
# Making the dataframe for the predicted values
predictedDf = df.copy()
predictedDf = predictedDf.dropna()
```

```
predictedDf = predictedDf[~predictedDf[0].isin([182,166])]

predictedDf['Accuracy'] = ((predictedDf[2].astype(int)) ==␣
 ↪predictedDf['Predicted'])
accuracy = sum(predictedDf['Accuracy']/len(predictedDf))
print('The accuracy of the spectral clustering algorithm is {:.2%}'.
 ↪format(accuracy))
```

```
The accuracy of the spectral clustering algorithm is 42.73%
```

[ ]:

HW1 Q4

May 24, 2020

[39]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[74]:
```
# Read in data
df = pd.read_csv("data/food-consumption.csv")
df.head()
```

[74]:

	Country	Real coffee	Instant coffee	Tea	Sweetener	Biscuits	\
0	Germany	90	49	88	19.0	57.0	
1	Italy	82	10	60	2.0	55.0	
2	France	88	42	63	4.0	76.0	
3	Holland	96	62	98	32.0	62.0	
4	Belgium	94	38	48	11.0	74.0	

	Powder soup	Tin soup	Potatoes	Frozen fish	…	Apples	Oranges	\
0	51	19	21	27	…	81	75	
1	41	3	2	4	…	67	71	
2	53	11	23	11	…	87	84	
3	67	43	7	14	…	83	89	
4	37	23	9	13	…	76	76	

	Tinned fruit	Jam	Garlic	Butter	Margarine	Olive oil	Yoghurt	\
0	44	71	22	91	85	74	30.0	
1	9	46	80	66	24	94	5.0	
2	40	45	88	94	47	36	57.0	
3	61	81	15	31	97	13	53.0	
4	42	57	29	84	80	83	20.0	

	Crisp bread
0	26
1	18
2	3
3	15
4	5

[5 rows x 21 columns]

[75]:
```
# Clean the data
df1 = df.copy()
removeRows = ['Sweden','Finland','Spain']

df1 = df1[~df1['Country'].isin(removeRows)].reset_index(drop=True)
df1
```

[75]:

	Country	Real coffee	Instant coffee	Tea	Sweetener	Biscuits	\
0	Germany	90	49	88	19.0	57.0	
1	Italy	82	10	60	2.0	55.0	
2	France	88	42	63	4.0	76.0	
3	Holland	96	62	98	32.0	62.0	
4	Belgium	94	38	48	11.0	74.0	
5	Luxembourg	97	61	86	28.0	79.0	
6	England	27	86	99	22.0	91.0	
7	Portugal	72	26	77	2.0	22.0	
8	Austria	55	31	61	15.0	29.0	
9	Switzerland	73	72	85	25.0	31.0	
10	Denmark	96	17	92	35.0	66.0	
11	Norway	92	17	83	13.0	62.0	
12	Ireland	30	52	99	11.0	80.0	

	Powder soup	Tin soup	Potatoes	Frozen fish	…	Apples	Oranges	\
0	51	19	21	27	…	81	75	
1	41	3	2	4	…	67	71	
2	53	11	23	11	…	87	84	
3	67	43	7	14	…	83	89	
4	37	23	9	13	…	76	76	
5	73	12	7	26	…	85	94	
6	55	76	17	20	…	76	68	
7	34	1	5	20	…	22	51	
8	33	1	5	15	…	49	42	
9	69	10	17	19	…	79	70	
10	32	17	11	51	…	81	72	
11	51	4	17	30	…	61	72	
12	75	18	2	5	…	57	52	

	Tinned fruit	Jam	Garlic	Butter	Margarine	Olive oil	Yoghurt	\
0	44	71	22	91	85	74	30.0	
1	9	46	80	66	24	94	5.0	
2	40	45	88	94	47	36	57.0	
3	61	81	15	31	97	13	53.0	
4	42	57	29	84	80	83	20.0	
5	83	20	91	94	94	84	31.0	
6	89	91	11	95	94	57	11.0	
7	8	16	89	65	78	92	6.0	
8	14	41	51	51	72	28	13.0	

```
9             46   61      64      82         48         61     48.0
10            50   64      11      92         91         30     11.0
11            34   51      11      63         94         28      2.0
12            46   89       5      97         25         31      3.0

    Crisp bread
0            26
1            18
2             3
3            15
4             5
5            24
6            28
7             9
8            11
9            30
10           34
11           62
12            9

[13 rows x 21 columns]
```

[76]:
```
# Checking if there is any missing data
df1.isna().any()
```

[76]:
```
Country           False
Real coffee       False
Instant coffee    False
Tea               False
Sweetener         False
Biscuits          False
Powder soup       False
Tin soup          False
Potatoes          False
Frozen fish       False
Frozen veggies    False
Apples            False
Oranges           False
Tinned fruit      False
Jam               False
Garlic            False
Butter            False
Margarine         False
Olive oil         False
Yoghurt           False
Crisp bread       False
dtype: bool
```

[77]:
```
columnNames = df1.columns.drop('Country')
```

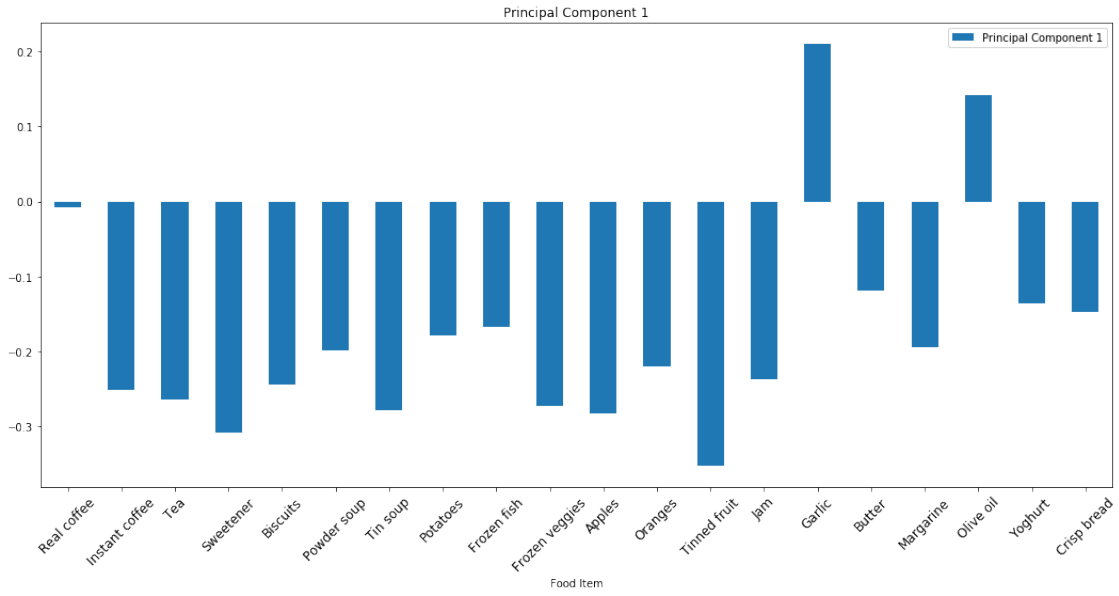
[78]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Separating out the features
x = df1.loc[:, columnNames].values
# Standardizing the features
x = StandardScaler().fit_transform(x)

pca = PCA(n_components=2)

# principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['Principal component 1', 'Principal component 2'])
```

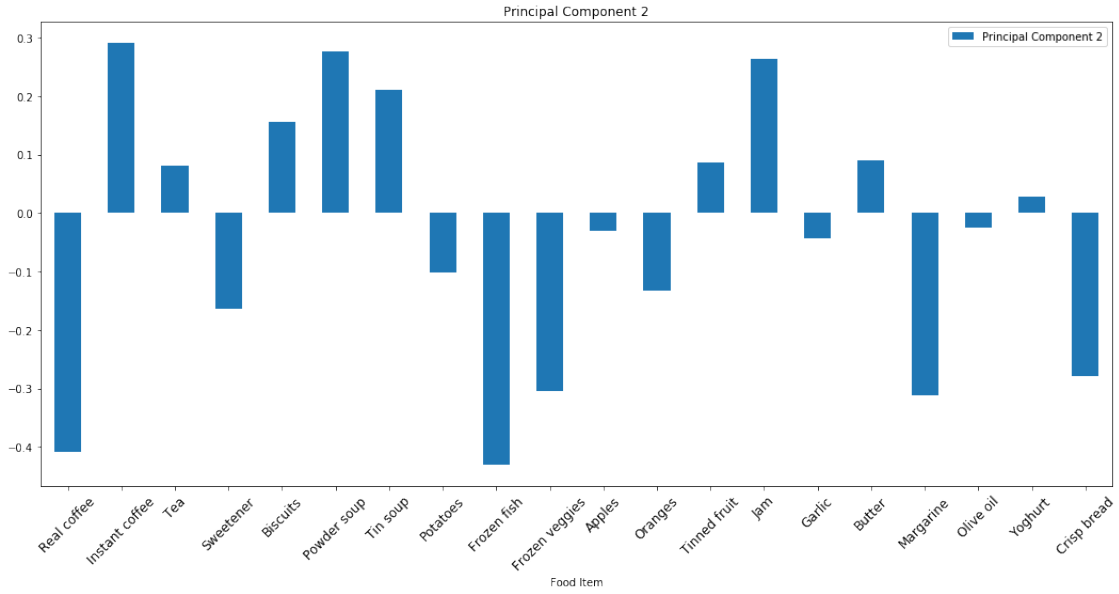
[79]:
```
pca.fit(x)
print("The principal components are {}.".format(pca.components_))
print("The explained variance is {}".format(pca.explained_variance_))
principalComponents = pca.fit_transform(x)
variance = pca.explained_variance_ratio_ #calculate variance ratios
print("The explained variance ratio for each component is {}".format(variance))
var=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=3)*100)
var #cumulative sum of variance explained with [n] features
print("The cumulative variance for the 2 components is {:.2f}%".format(var[1]))
```

```
The principal components are [[-0.00783844 -0.25088295 -0.26379551 -0.3075616
-0.24333708 -0.19829443
  -0.27740544 -0.17786406 -0.16708899 -0.27206292 -0.28289229 -0.219254
  -0.35276092 -0.2361455   0.20987329 -0.11823783 -0.19339383  0.14145774
  -0.13612727 -0.14655957]
 [-0.4084629   0.29072317  0.08012012 -0.16292522  0.15635054  0.27659622
   0.21002179 -0.1021976  -0.43127954 -0.30435236 -0.03059694 -0.13242838
   0.08723895  0.26308945 -0.04367603  0.09016861 -0.31240199 -0.02485444
   0.02732938 -0.27792015]].
The explained variance is [7.27704171 3.62085054]
The explained variance ratio for each component is [0.33586346 0.16711618]
The cumulative variance for the 2 components is 50.30%
```

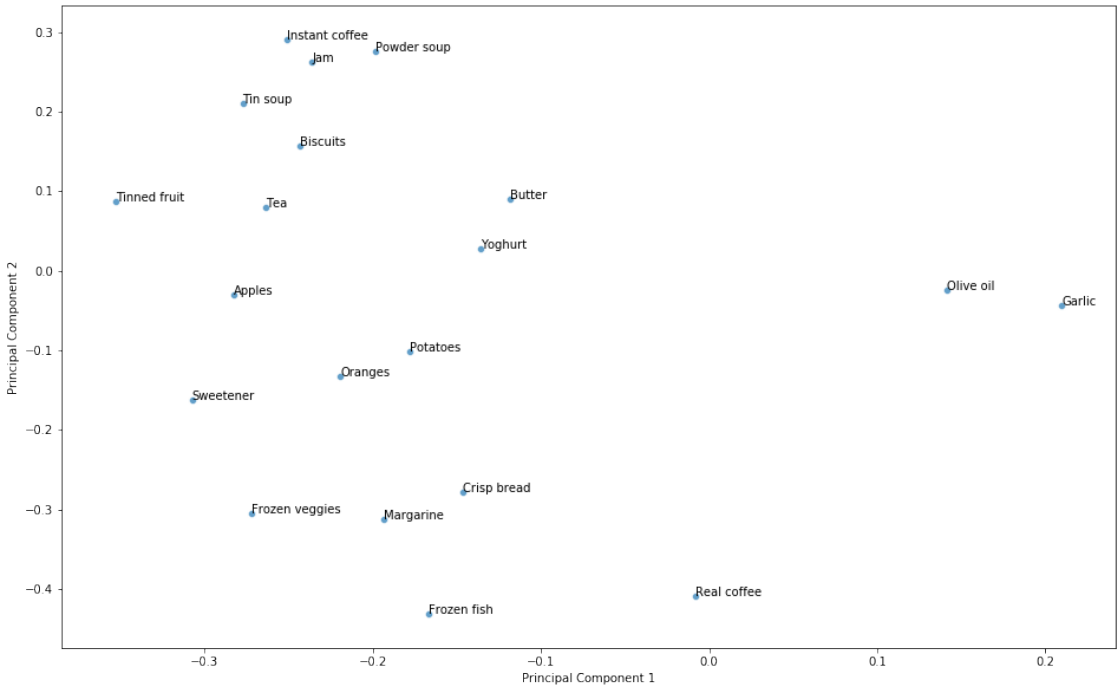
[80]:
```
df = pd.DataFrame(pca.components_,columns=df1.columns[1:],index = ['Principal␣
 ↪Component 1','Principal Component 2'])
df= df.transpose()
df['Food Item'] = df.index

ax = df.plot.bar(x='Food Item', y='Principal Component 1', rot=0,figsize =␣
 ↪(18,8))
plt.title("Principal Component 1")
```

```
plt.xticks(fontsize=12, rotation=45)
plt.show()

ax = df.plot.bar(x='Food Item', y='Principal Component 2', rot=0,figsize =␣
↪(18,8))
plt.title("Principal Component 2")
plt.xticks(fontsize=12, rotation=45)
plt.show()
```



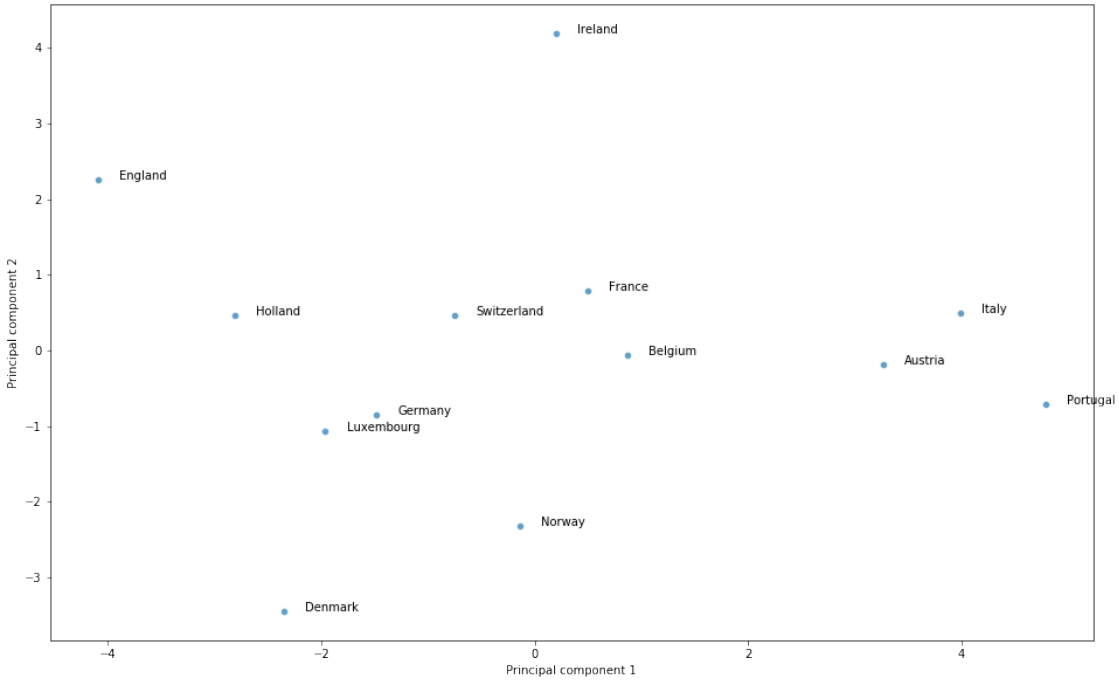


[81]:
```
components = pd.DataFrame(pca.components_.T)
components = components.rename ({0:"Principal Component 1",1:"Principal␣
↪Component 2"}, axis = 'columns')
plt.figure(figsize=(16,10))
p1 = sns.scatterplot(
    x="Principal Component 1", y="Principal Component 2",
    palette=sns.color_palette("hls", 10),
    data=components,
    legend="full",
    alpha=0.7
)
foods = df1.columns.values
foods = np.delete(foods,0)

for i in range(0,len(foods)):
    p1.text(components['Principal Component 1'][i],
            components['Principal Component 2'][i],
            foods[i])
```



[82]:
```
import seaborn as sns
plt.figure(figsize=(16,10))
p1 = sns.scatterplot(
    x="Principal component 1", y="Principal component 2",
    palette=sns.color_palette("hls", 10),
    data=principalDf,
```

```
    legend="full",
    alpha=0.7
)

for i in range(0,len(df1)):
    p1.text(principalDf['Principal component 1'][i]+0.2,
            principalDf['Principal component 2'][i],
            df1['Country'][i])
```



[ ]: