Model unsupervised and viz

May 12, 2020

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from os import listdir
from os.path import isfile, join
import re
from matplotlib import pyplot as plt
```

[3]:
```
def isjpg(filepath):
    return re.search(".jpg$", filepath)
```

[4]:
```
# Function Section
def calculate_pad(brightness, saturation):
    p = 0.69*brightness + 0.22*saturation
    a = -0.31*brightness + 0.6*saturation
    d = 0.76*brightness + 0.32*saturation
    return [p,d,a]

def calculate_blur(img):
    return cv2.Laplacian(img, cv2.CV_64F).var()

def mean_brightness(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) #convert it to hsv
    return np.mean(hsv[:,:,2])

def mean_saturation(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) #convert it to hsv
    return np.mean(hsv[:,:,1])

def calculate_opticalFlow(img1, img2):
    f, axarr = plt.subplots(2,1)
    axarr[0].imshow(img1)
    axarr[1].imshow(img2)
    plt.show()
    prev = cv2.cvtColor(testEld[0], cv2.COLOR_BGR2GRAY)
    forward = cv2.cvtColor(testEld[1], cv2.COLOR_BGR2GRAY)
```

```
    mask = np.zeros_like(prev)
    mask[..., 1] = 255
    flow = cv2.calcOpticalFlowFarneback(prev, forward, flow=None, pyr_scale=0.
 ↪5,levels =1,winsize=3,iterations=15,poly_n=3,poly_sigma=5,flags=cv2.
 ↪OPTFLOW_FARNEBACK_GAUSSIAN)
    magnitude, angle = cv2.cartToPolar(flow[..., 0], flow[..., 1])
    return cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX)[0]
```

[5]:
```
ANGER_POINT = [-0.43, 0.67, 0.34]
JOY_POINT = [0.76, 0.48, 0.35]
SURPRISE_POINT = [0.4, 0.67, -0.13]
DISGUST_POINT = [-0.6, 0.35, 0.11]
FEAR_POINT = [-0.64, 0.6, -0.43]
SADNESS_POINT = [-0.63, 0.27, -0.33]


from scipy import spatial
kdtree = spatial.cKDTree(np.
 ↪array([ANGER_POINT,JOY_POINT,SURPRISE_POINT,DISGUST_POINT,FEAR_POINT,SADNESS_POINT]))
```

[6]:
```
def label(x):
    try:
        dist, ix = kdtree.query(x,k=1)
    except Exception as e:
        print(x)
        print(e)
    if ix==0:
        return 'Anger'
    elif ix==1:
        return "Joy"
    elif ix==2:
        return 'Surprise'
    elif ix==-3:
        return "Disgust"
    elif ix==4:
        return "Fear"
    else:
        return 'Sadness'
```

[7]:
```
import glob
import re
from scipy.interpolate import interp1d
```

[8]:
```
modeldf=pd.read_csv('clean_df.csv')
```

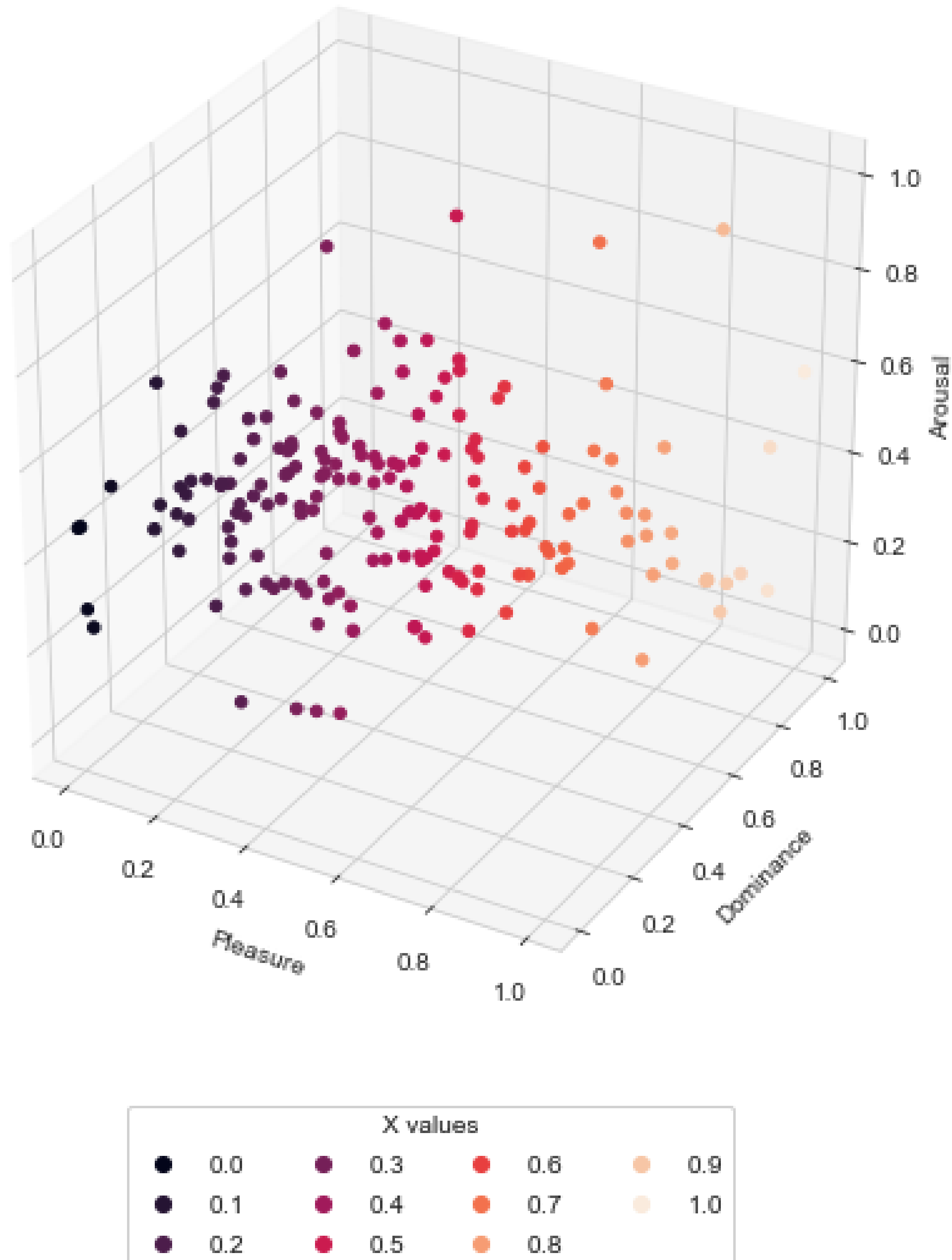
[9]:
```
tpdf=modeldf[['scene_avg_p',␣
 ↪'scene_avg_a','scene_avg_d','scene_avg_blur','scene_avg_optical_flow']]
```

[10]:
```
#normallization
modeldf[['scene_avg_p',␣
↪'scene_avg_a','scene_avg_d','scene_avg_blur','scene_avg_optical_flow']]=(tpdf-tpdf.
↪min())/(tpdf.max()-tpdf.min())
```

2. 3D visualization of scenes'PAD

[11]:
```
import seaborn as sns, numpy as np, pandas as pd, random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
sns.set_style("whitegrid", {'axes.grid' : False})

fig = plt.figure(figsize=(6,6))

ax = Axes3D(fig)

x = modeldf['scene_avg_p'].tolist()
y = modeldf['scene_avg_d'].tolist()
z = modeldf['scene_avg_a'].tolist()

cm = sns.palplot(sns.color_palette("BrBG", 7))
g = ax.scatter(x, y, z, c=x, marker='o', depthshade=False, cmap=cm)
ax.set_xlabel('Pleasure')
ax.set_ylabel('Dominance')
ax.set_zlabel('Arousal')

# produce a legend with the unique colors from the scatter
legend = ax.legend(*g.legend_elements(), loc="lower center", title="X values",␣
↪borderaxespad=-10, ncol=4)
ax.add_artist(legend)

plt.show()
```

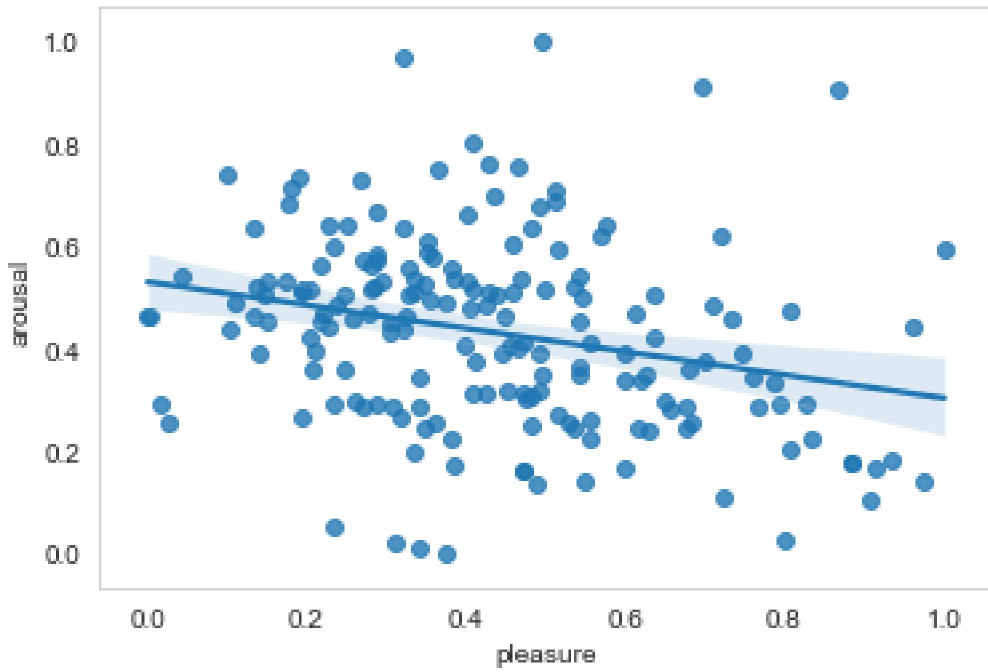
```
C:\Users\YihengYe\Anaconda3\lib\site-packages\statsmodels\tools\_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```


Pleasure
Dominance
Arousal
0.0
0.2
0.4
0.6
0.8
1.0
X values
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

3. Other Viz

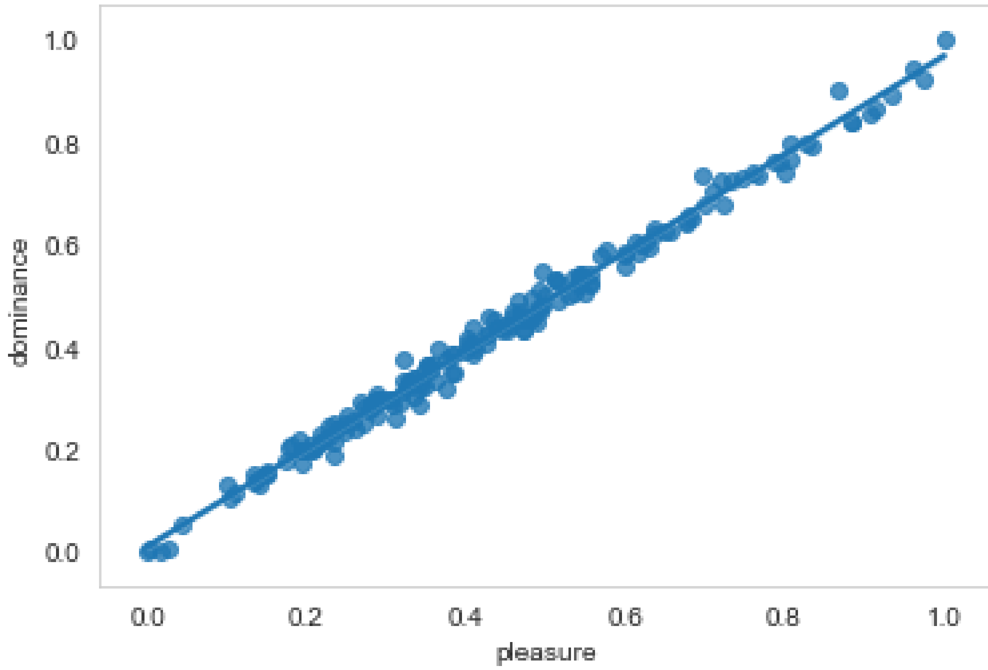
[12]:
```
import seaborn as sns
sns.regplot(modeldf['scene_avg_p'],modeldf['scene_avg_a'])
plt.xlabel('pleasure')
plt.ylabel('arousal')
```

[12]: Text(0, 0.5, 'arousal')



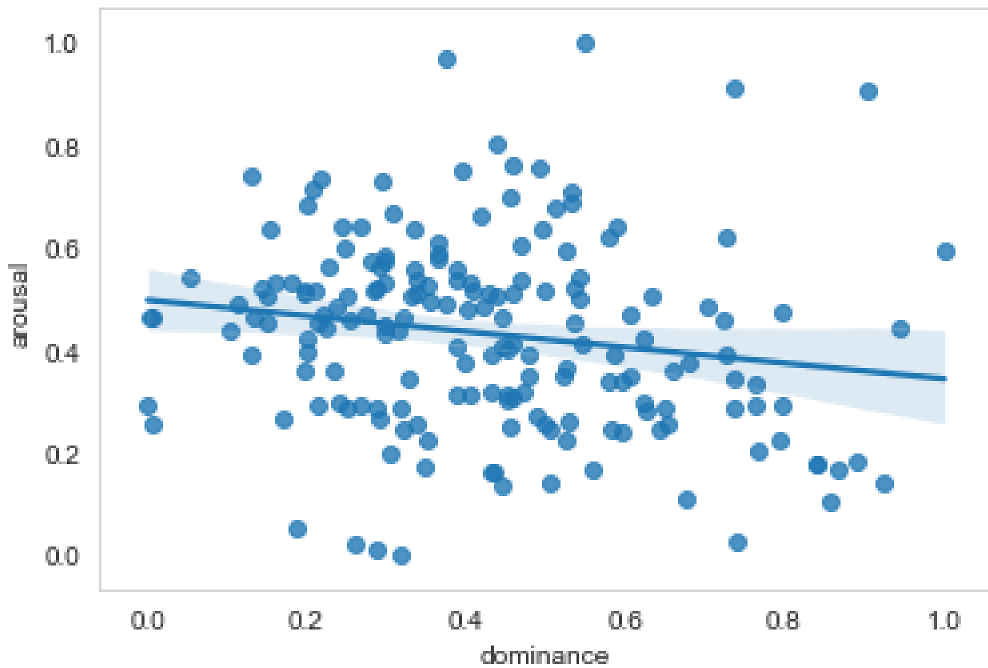
[13]:
```
sns.regplot(modeldf['scene_avg_p'],modeldf['scene_avg_d'])
plt.xlabel('pleasure')
plt.ylabel('dominance')
```

[13]: Text(0, 0.5, 'dominance')



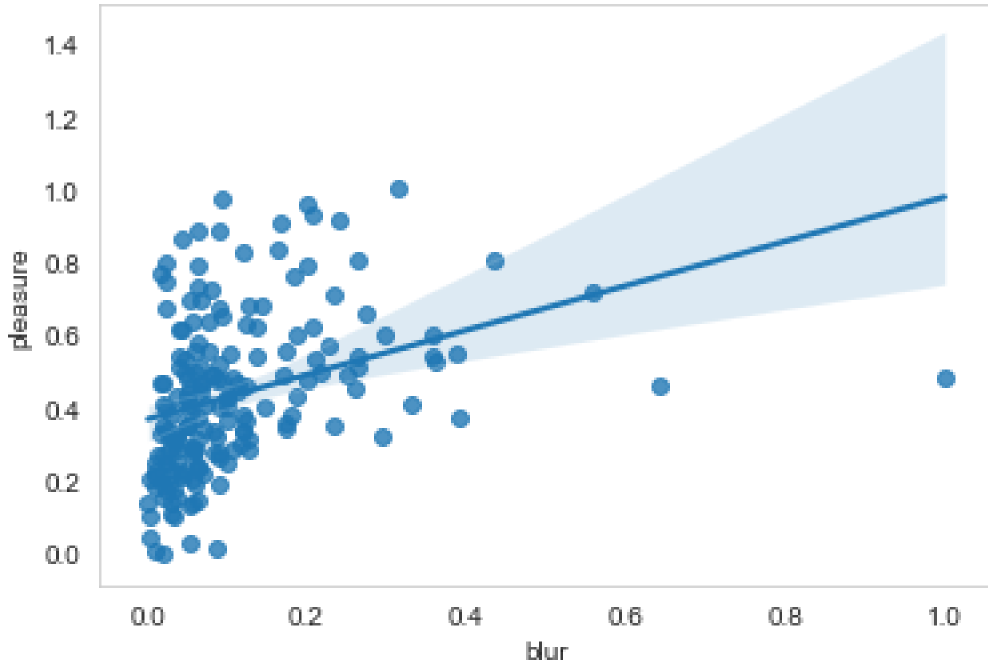
[14]:
```
sns.regplot(modeldf['scene_avg_d'],modeldf['scene_avg_a'])
plt.xlabel('dominance')
plt.ylabel('arousal')
```

[14]: `Text(0, 0.5, 'arousal')`



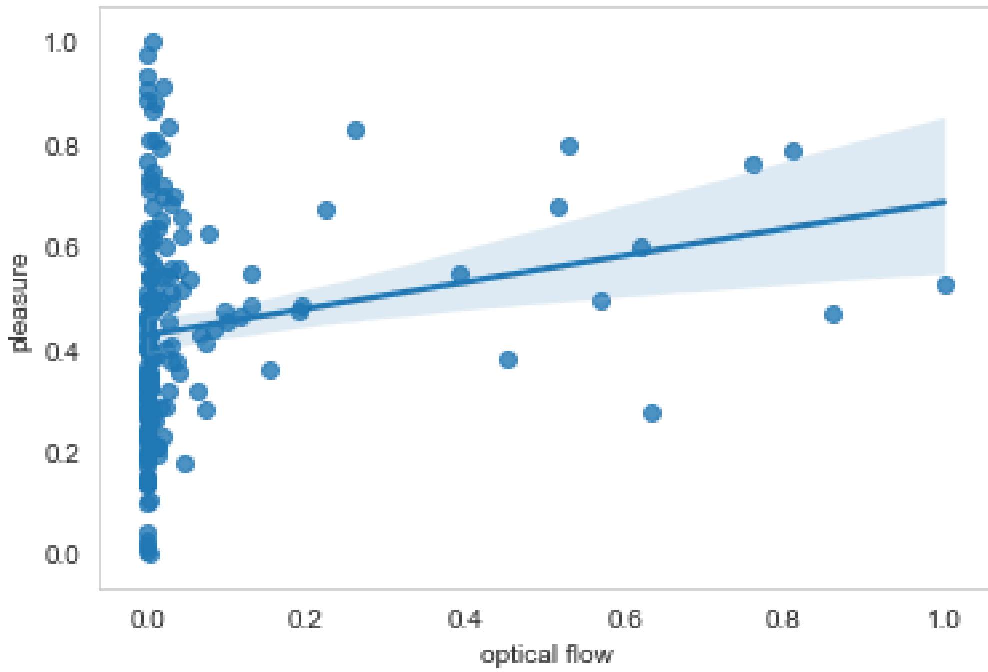
[15]:
```
sns.regplot(modeldf['scene_avg_blur'],modeldf['scene_avg_p'])
plt.xlabel('blur')
plt.ylabel('pleasure')
```

[15]: Text(0, 0.5, 'pleasure')



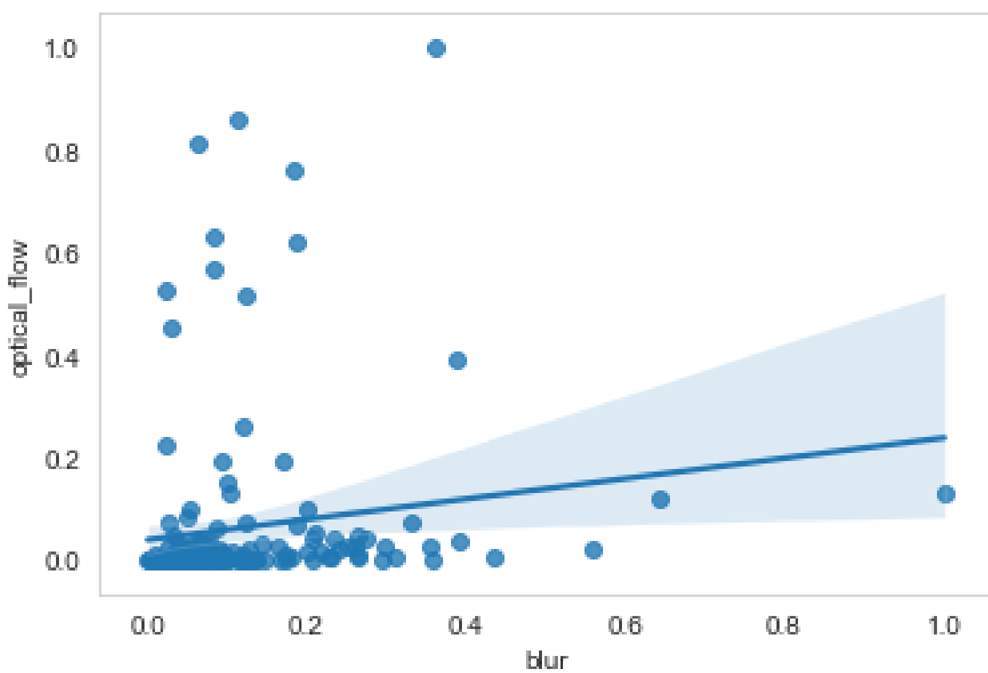
[16]:
```
sns.regplot(modeldf['scene_avg_optical_flow'],modeldf['scene_avg_p'])
plt.xlabel('optical flow')
plt.ylabel('pleasure')
```

[16]: Text(0, 0.5, 'pleasure')




```
[17]: sns.regplot(modeldf['scene_avg_blur'],modeldf['scene_avg_optical_flow'])
plt.xlabel('blur')
plt.ylabel('optical_flow')
```

```
[17]: Text(0, 0.5, 'optical_flow')
```



4. Cluster

We build 6 clusters here to classifer our emotion

[18]:
```
from sklearn.cluster import KMeans
```

[19]:
```
numdf=modeldf[['scene_avg_p',␣
↪'scene_avg_a','scene_avg_d','scene_avg_blur','scene_avg_optical_flow']]
```

[20]:
```
numdf=numdf.fillna(0)
```

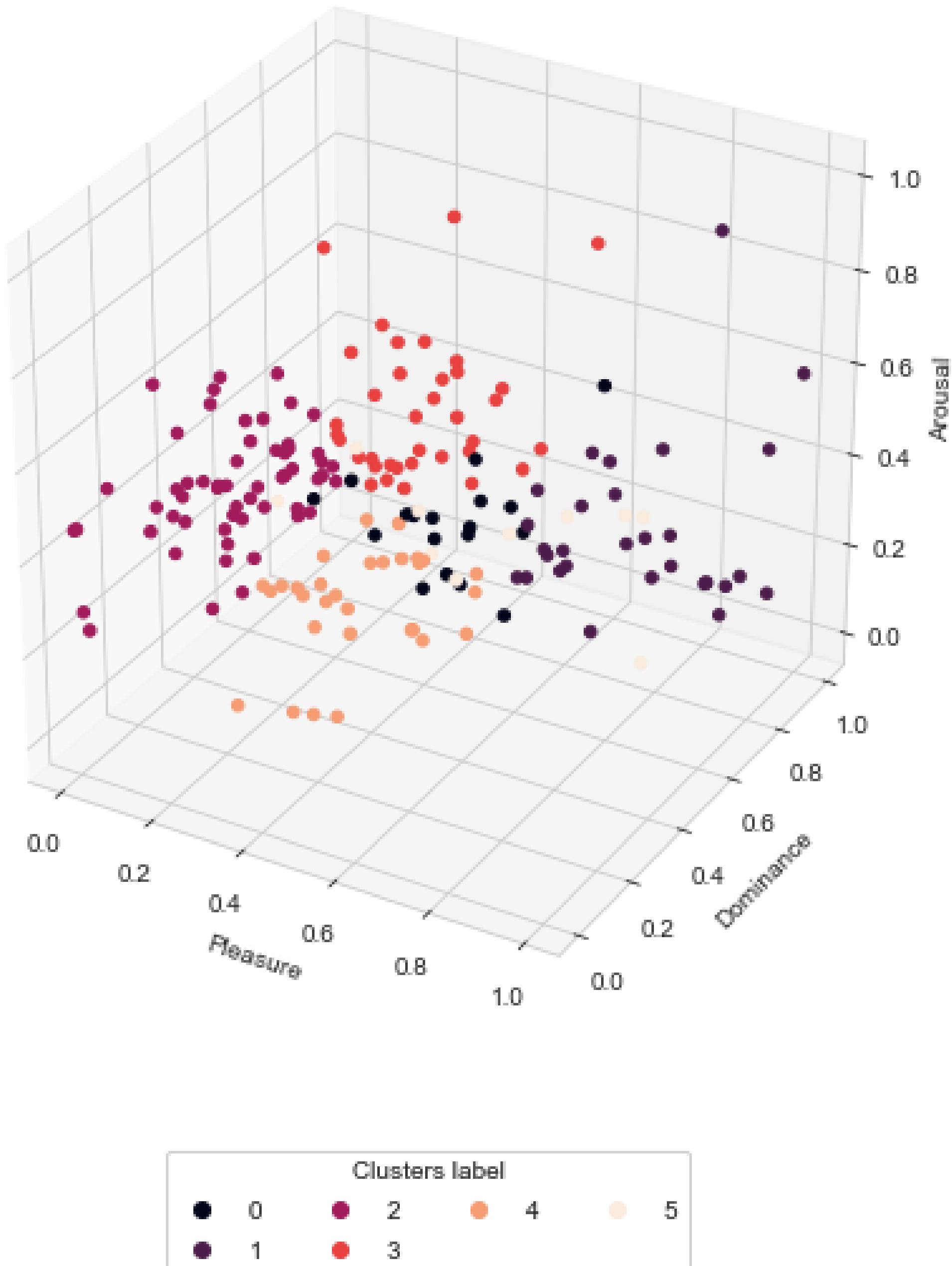
[21]:
```
kmeans=KMeans(n_clusters=6, random_state=0)
```

[22]:
```
kmeans=kmeans.fit(numdf)
```

[23]:
```
modeldf['emotion_clusters']=kmeans.labels_
```

Plot the 3d graph again, with clusters colors

[24]:
```
sns.set_style("whitegrid", {'axes.grid' : False})

fig = plt.figure(figsize=(6,6))

ax = Axes3D(fig)

x = modeldf['scene_avg_p'].tolist()
y = modeldf['scene_avg_d'].tolist()
z = modeldf['scene_avg_a'].tolist()
ec=modeldf['emotion_clusters'].tolist()

cm = sns.palplot(sns.color_palette("BrBG", 7))
g = ax.scatter(x, y, z, c=ec, marker='o', depthshade=False, cmap=cm)
ax.set_xlabel('Pleasure')
ax.set_ylabel('Dominance')
ax.set_zlabel('Arousal')

# produce a legend with the unique colors from the scatter
legend = ax.legend(*g.legend_elements(), loc="lower center", title="Clusters␣
↪label", borderaxespad=-10, ncol=4)
ax.add_artist(legend)

plt.show()
```


Pleasure
0.0
0.2
0.4
0.6
0.8
1.0
Dominance
0.0
0.2
0.4
0.6
0.8
1.0
Arousal
0.0
0.2
0.4
0.6
0.8
1.0
Clusters label
0
1
2
3
4
5

This looks quite different than the PAD model only as the 6 clusters are clearly spreaded out

4. Further improvement: Dimension Reduction

1) t-SNE

[32]:
```
from sklearn.manifold import TSNE
```

[33]:
```
X_embeded=TSNE(n_components=2, random_state=0).fit_transform(numdf)
```

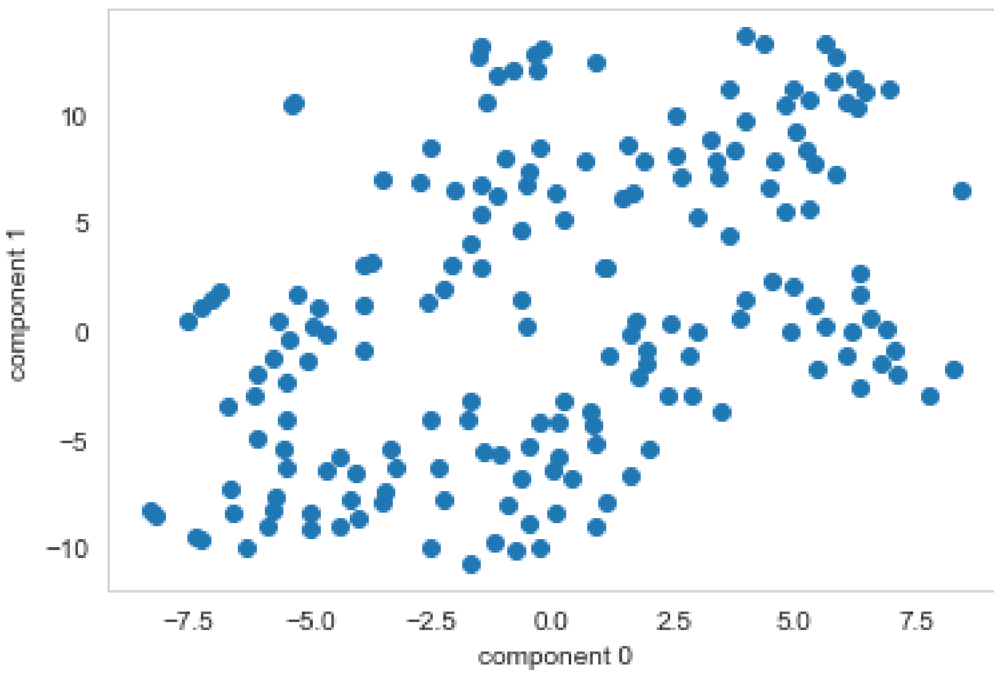
[34]:
```
kmeans2=KMeans(n_clusters=6).fit(X_embeded)
```

[35]:
```
dftsne=pd.DataFrame(X_embeded)
```

[36]:
```
dftsne['emotion_clusters']=kmeans2.labels_
```

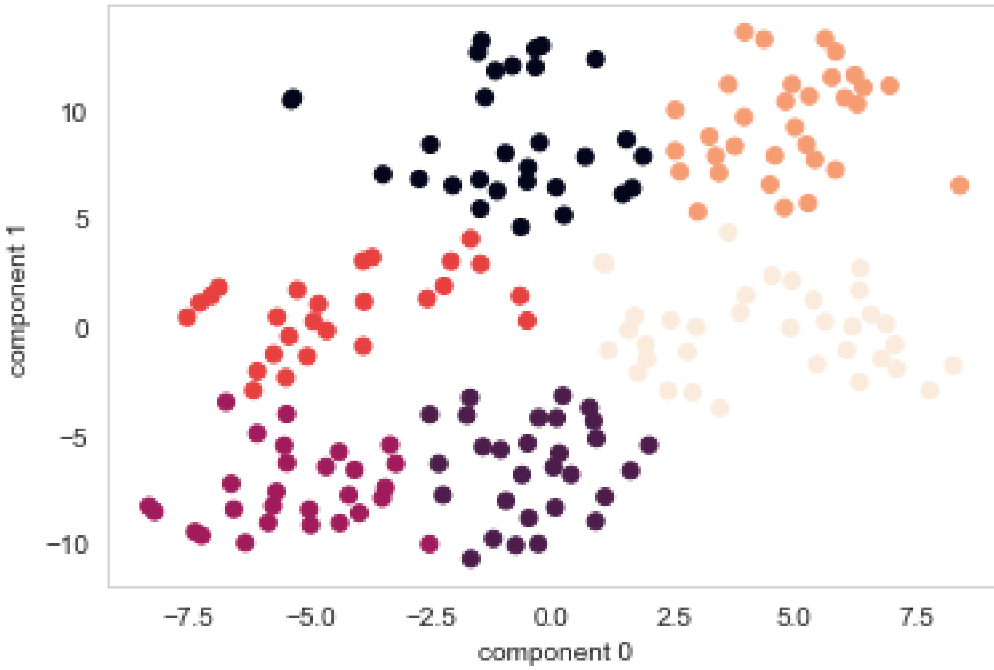
[37]:
```
#two components
plt.scatter(dftsne[0], dftsne[1])
plt.xlabel('component 0')
plt.ylabel('component 1')
```

[37]: `Text(0, 0.5, 'component 1')`



[38]:
```
#6 clusters
plt.scatter(dftsne[0], dftsne[1], c=dftsne['emotion_clusters'])
plt.xlabel('component 0')
plt.ylabel('component 1')
```

[38]: Text(0, 0.5, 'component 1')

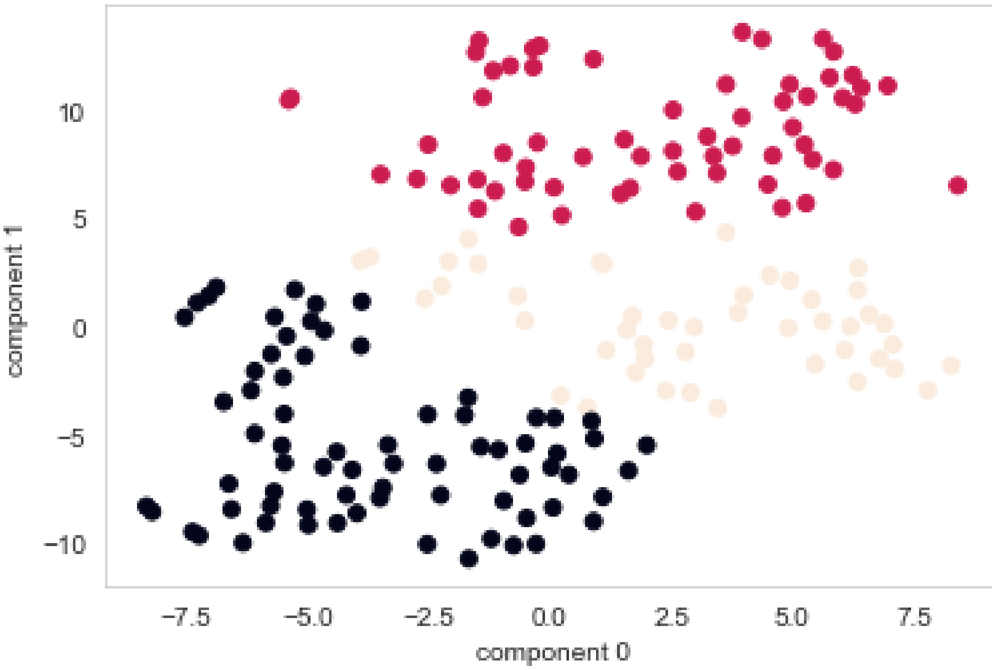


According to TSNE vis, we can have 3 emotions instead of 6, which will be better

[43]:
```
kmeans3=KMeans(n_clusters=3).fit(X_embeded)
dftsne['cluster2']=kmeans3.labels_
```

[44]:
```
#3 clusters
plt.scatter(dftsne[0], dftsne[1], c=dftsne['cluster2'])
plt.xlabel('component 0')
plt.ylabel('component 1')
```

[44]: Text(0, 0.5, 'component 1')



2.) PCA

[45]:
```
from sklearn.decomposition import PCA
```

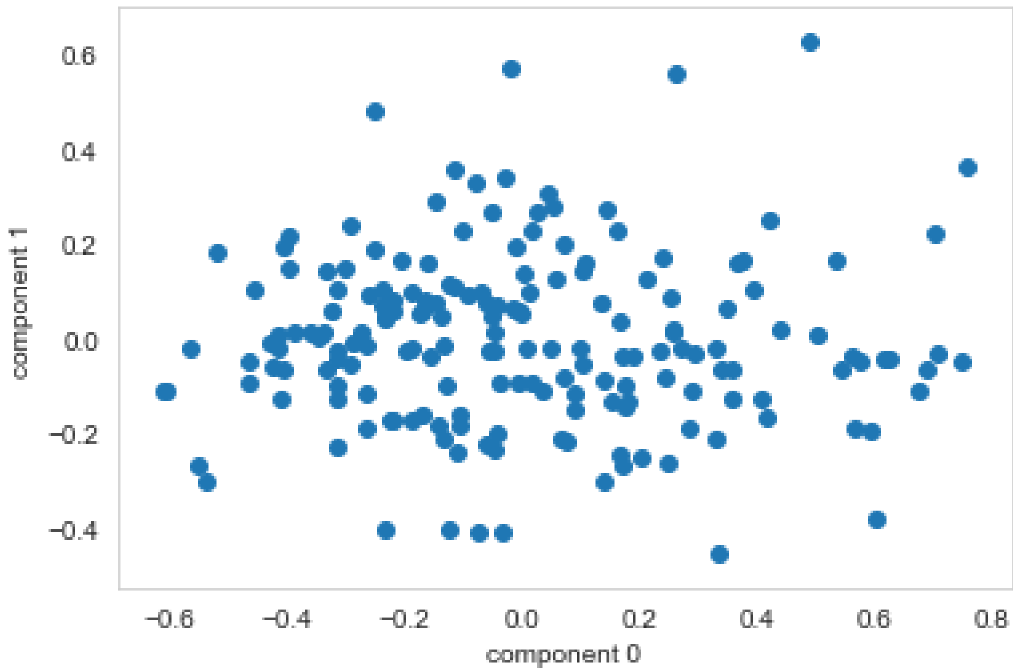
[46]:
```
pca=PCA(n_components=2)
```

[47]:
```
pca=pca.fit(numdf)
```

[48]:
```
dfpca=pd.DataFrame(pca.transform(numdf))
```

[49]:
```
plt.scatter(dfpca[0], dfpca[1])
plt.xlabel('component 0')
plt.ylabel('component 1')
```

[49]:
```
Text(0, 0.5, 'component 1')
```

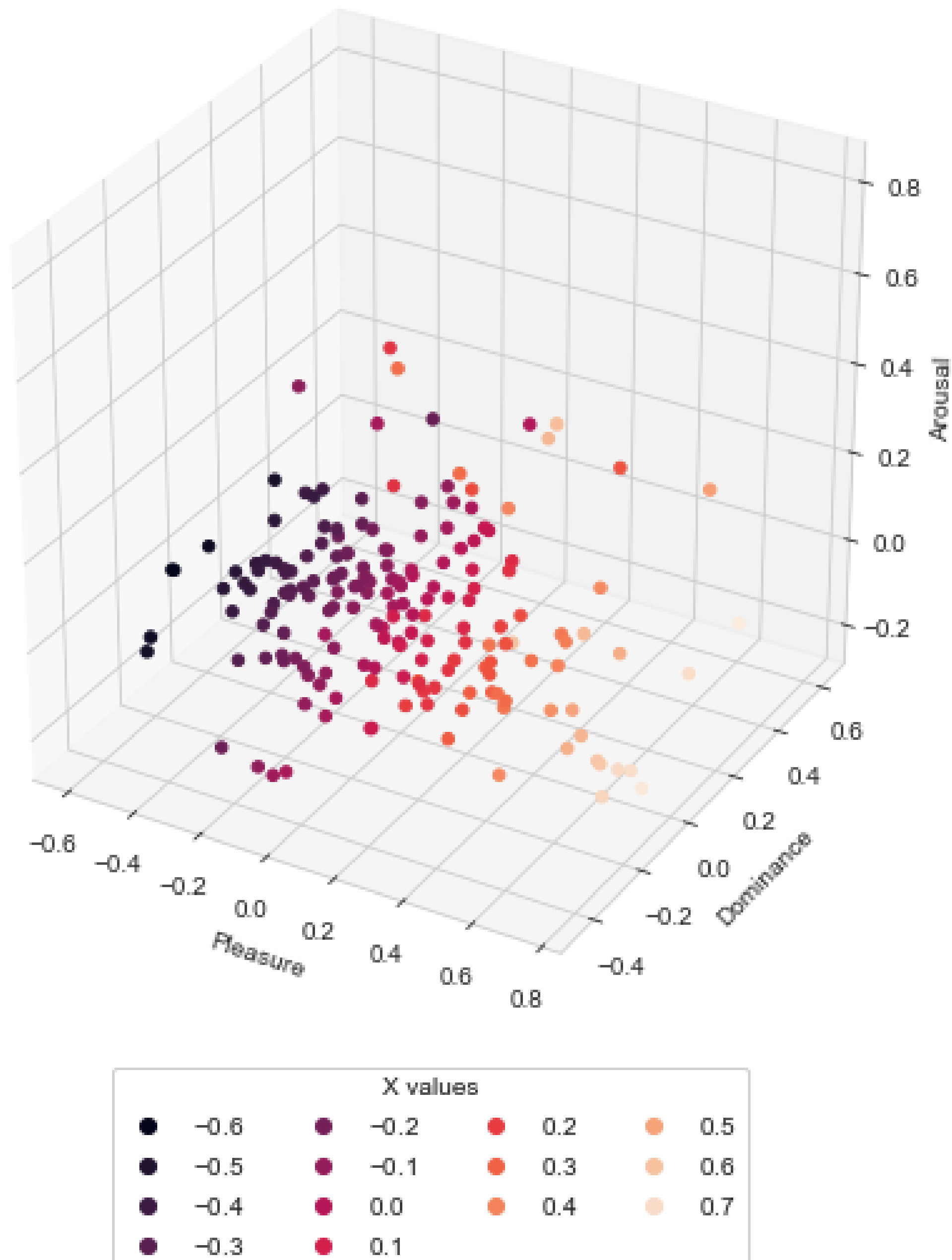


it looks terrible for clustering, we may need more components

[50]:
```
dfpca2=pd.DataFrame(PCA(n_components=3).fit_transform(numdf))
```

[51]:
```
sns.set_style("whitegrid", {'axes.grid' : False})

fig = plt.figure(figsize=(6,6))

ax = Axes3D(fig)

x = dfpca2[0].tolist()
y = dfpca2[1].tolist()
z = dfpca2[2].tolist()


cm = sns.palplot(sns.color_palette("BrBG", 7))
g = ax.scatter(x, y, z, c=x, marker='o', depthshade=False, cmap=cm)
ax.set_xlabel('Pleasure')
ax.set_ylabel('Dominance')
ax.set_zlabel('Arousal')

# produce a legend with the unique colors from the scatter
legend = ax.legend(*g.legend_elements(), loc="lower center", title="X values",␣
 ↪borderaxespad=-10, ncol=4)
ax.add_artist(legend)
```

```
plt.show()
```



0.8
0.6
0.4
0.2
0.0
−0.2
Arousal
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
Pleasure
−0.4
−0.2
0.0
0.2
0.4
0.6
Dominance
X values
−0.6
−0.5
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

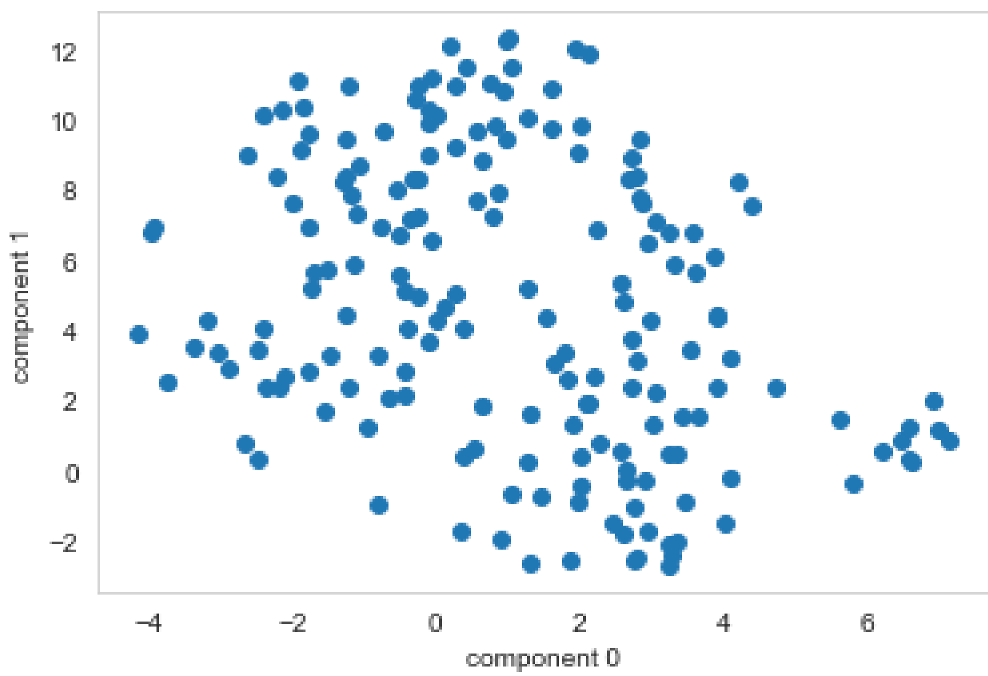
still not very ideal

3. tuning t-SNE

We need to tune t-SNE to look good on our datasets as PCA's performance is too bad

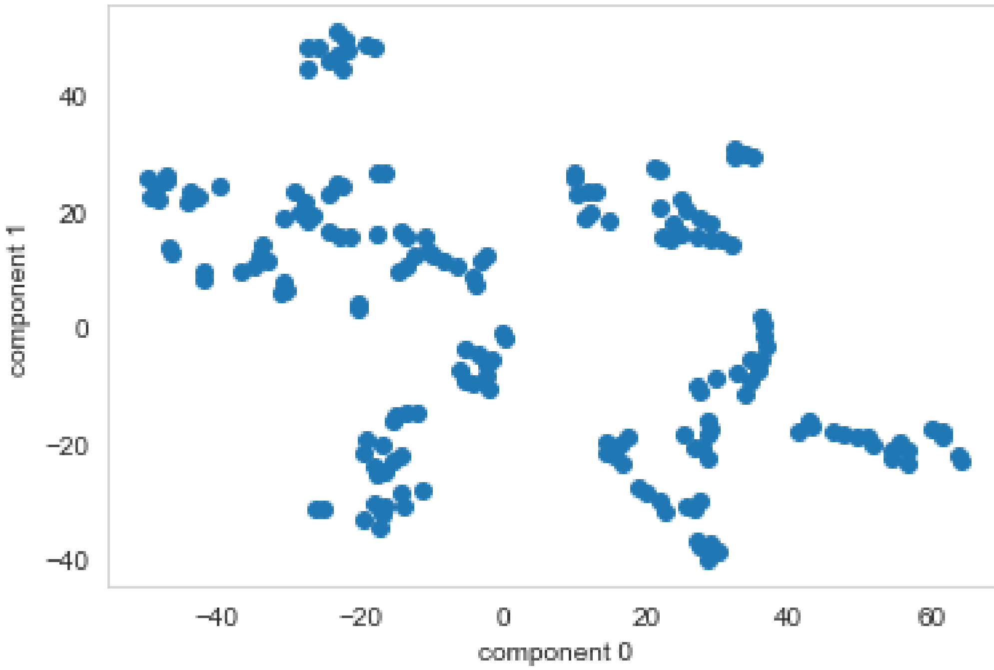
[55]:
```
tsne2=TSNE(n_components=2, random_state=0, perplexity=50)
dftsn2=pd.DataFrame(tsne2.fit_transform(numdf))
plt.scatter(dftsn2[0], dftsn2[1])
plt.xlabel('component 0')
plt.ylabel('component 1')
```

[55]:
```
Text(0, 0.5, 'component 1')
```



[56]:
```
tsne3=TSNE(n_components=2, random_state=0, perplexity=5)
dftsn3=pd.DataFrame(tsne3.fit_transform(numdf))
plt.scatter(dftsn3[0], dftsn3[1])
plt.xlabel('component 0')
plt.ylabel('component 1')
```

[56]: Text(0, 0.5, 'component 1')



Clearly, when perplexity=5, we have a nice scatter plots with a Hexagonal star each representing an emotion.

[ ]: