2023-05-15_ChartTypes_021_Chernoff

June 5, 2023

[23]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 Chernoff faces

1.1 Function for drawing

[26]:
```
def cface(ax, posX,posY,s,␣
 ↪x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18):
    # based on:
    # https://healthyalgorithms.com/2012/11/12/
 ↪dataviz-in-python-chernoff-faces-with-matplotlib/
    # https://gist.github.com/aflaxman/4043086

    # posX, posY: position of center of face
    # s: scale of face
    tf=matplotlib.transforms.Affine2D(matrix=np.array([[s,0.,posX],[0.
 ↪,s,posY],[0.,0.,1.]]))+ax.transData

    # x1 = height  of upper face
    # x2 = overlap of lower face
    # x3 = half of vertical size of face
    # x4 = width of upper face
    # x5 = width of lower face
```

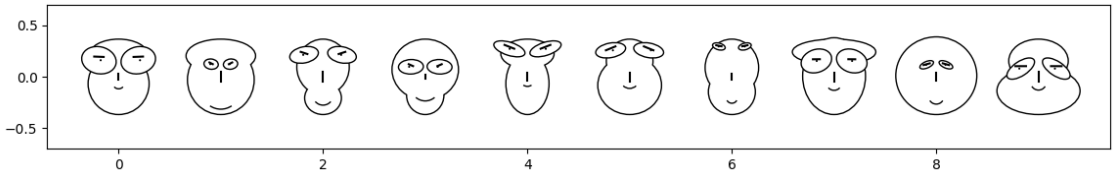
```
    # x6 = length of nose
    # x7 = vertical position of mouth
    # x8 = curvature of mouth
    # x9 = width of mouth
    # x10 = vertical position of eyes
    # x11 = separation of eyes
    # x12 = slant of eyes
    # x13 = eccentricity of eyes
    # x14 = size of eyes
    # x15 = position of pupils
    # x16 = vertical position of eyebrows
    # x17 = slant of eyebrows
    # x18 = size of eyebrows

    # transform some values so that input between 0,1 yields variety of output
    x3 = 1.9*(x3-.5)
    x4 = (x4+.25)
    x5 = (x5+.2)
    x6 = .3*(x6+.01)
    x8 = 5*(x8+.001)
    x11 /= 5
    x12 = 2*(x12-.5)
    x13 += .05
    x14 += .1
    x15 = .5*(x15-.5)
    x16 = .25*x16
    x17 = .5*(x17-.5)
    x18 = .5*(x18+.1)

    # top of face, in box with l=-x4, r=x4, t=x1, b=x3
    e = matplotlib.patches.Ellipse( (0,(x1+x3)/2), 2*x4, (x1-x3), fc='white',
            linewidth=2,ec="k",transform=tf)
    ax.add_artist(e)

    # bottom of face, in box with l=-x5, r=x5, b=-x1, t=x2+x3
    e = matplotlib.patches.Ellipse( (0,(-x1+x2+x3)/2), 2*x5, (x1+x2+x3),␣
↪fc='white',
            linewidth=2,ec="k",transform=tf)
    ax.add_artist(e)

    # cover overlaps
    e = matplotlib.patches.Ellipse( (0,(x1+x3)/2), 2*x4, (x1-x3), fc='white',␣
↪ec='none',transform=tf)
    ax.add_artist(e)
    e = matplotlib.patches.Ellipse( (0,(-x1+x2+x3)/2), 2*x5, (x1+x2+x3),␣
↪fc='white', ec='none',transform=tf)
    ax.add_artist(e)
```

```
    # draw nose
    ax.plot([0,0], [-x6/2, x6/2], 'k',transform=tf)

    # draw mouth
    p = matplotlib.patches.Arc( (0,-x7+.5/x8), 1/x8, 1/x8, theta1=270-180/np.
↪pi*np.arctan(x8*x9),
              theta2=270+180/np.pi*np.arctan(x8*x9),color="k",transform=tf)
    ax.add_artist(p)

    # draw eyes
    p = matplotlib.patches.Ellipse( (-x11-x14/2,x10), x14, x13*x14, angle=-180/
↪np.pi*x12,
              facecolor='white',ec="k",transform=tf)
    ax.add_artist(p)

    p = matplotlib.patches.Ellipse( (x11+x14/2,x10), x14, x13*x14, angle=180/np.
↪pi*x12,
              facecolor='white',ec="k",transform=tf)
    ax.add_artist(p)

    # draw pupils
    p = matplotlib.patches.Ellipse( (-x11-x14/2-x15*x14/2, x10), .05, .05,␣
↪facecolor='black',transform=tf)
    ax.add_artist(p)
    p = matplotlib.patches.Ellipse( (x11+x14/2-x15*x14/2, x10), .05, .05,␣
↪facecolor='black',transform=tf)
    ax.add_artist(p)

    # draw eyebrows
    ax.plot([-x11-x14/2-x14*x18/2,-x11-x14/2+x14*x18/
↪2],[x10+x13*x14*(x16+x17),x10+x13*x14*(x16-x17)],
              'k',transform=tf)
    ax.plot([x11+x14/2+x14*x18/2,x11+x14/2-x14*x18/
↪2],[x10+x13*x14*(x16+x17),x10+x13*x14*(x16-x17)],
              'k',transform=tf)
```

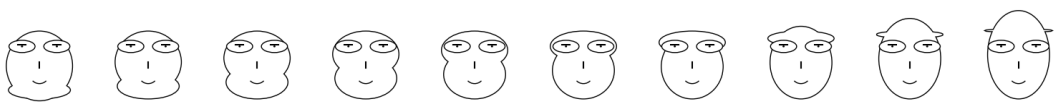
1.2 A few single faces to get acquainted

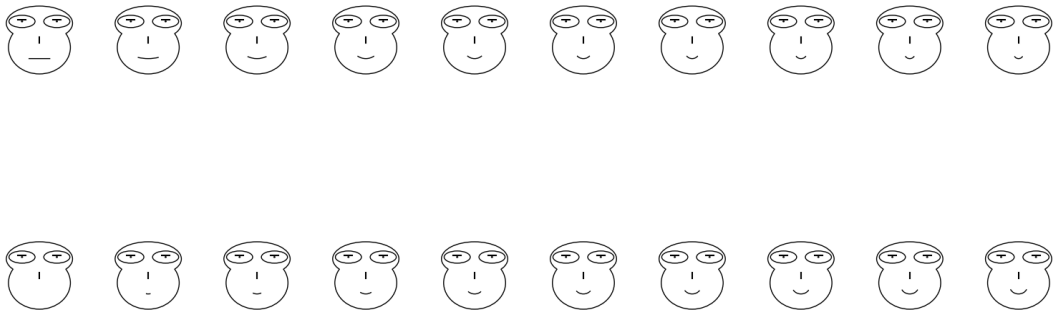
[25]:
```
# draw a random sequence
iMax=10
buffer=0.7
fig=plt.figure(figsize=(2*iMax,2))
ax=fig.add_subplot(aspect=1.)
for i in range(iMax):
```

```
    x=0.2+np.random.random(17)*0.6
    cface(ax,i,0.,0.4, .9, *x)
ax.axis([-buffer,iMax-1+buffer,-buffer,buffer])
plt.tight_layout()
plt.show()
```



[27]:
```
# vary one parameter at a time
for parNr in [0,1,2,6,7]:
    iMax=10
    buffer=0.6
    zList=np.linspace(0.,1.,num=iMax)
    fig=plt.figure(figsize=(2*iMax,2))
    ax=fig.add_subplot()
    for i in range(iMax):
        x=np.full(17,0.5)
        x[parNr]=zList[i]
        cface(ax,i,0.,0.4, .9, *x)
    ax.axis([-buffer,iMax-1+buffer,-buffer,buffer])
    plt.axis("off")
    plt.show()
```



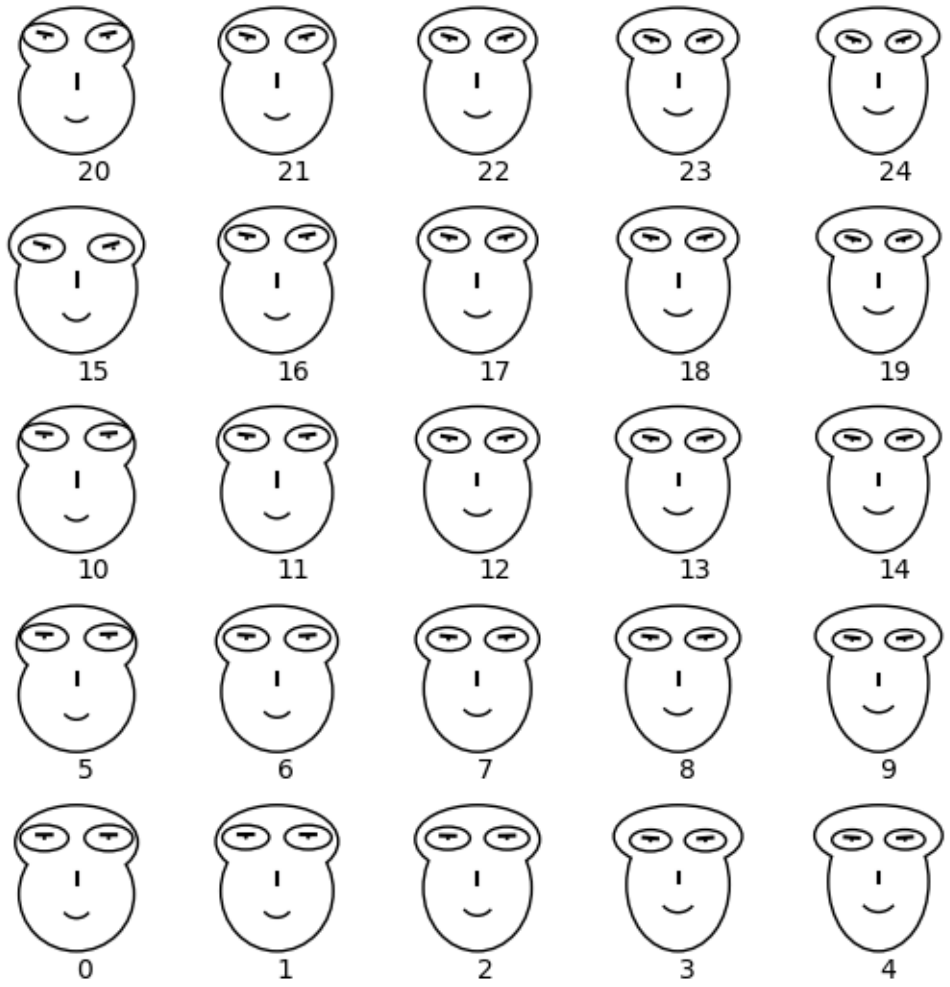


1.3 Try to spot outliers

[28]:
```
# create nPts * nPts grid of faces with random parameters
nPts=5

# create prep data
dataPre=np.zeros((nPts,nPts,3))
# positions
dataPre[:,:,0]=np.arange(nPts).reshape((1,-1))/nPts
dataPre[:,:,1]=np.arange(nPts).reshape((-1,1))/nPts
# a bit of randomness
dataPre[:,:,2]=0.2*np.random.random(size=(nPts,nPts))
dataPre=dataPre.reshape((-1,3))

# create full data
data=np.zeros((nPts*nPts,3+18))
# copy positions
data[:,:2]=dataPre[:,:2]
sizeparam=.4
sizeparam2=0.6
# size and x1
data[:,2]=sizeparam/nPts
data[:,3]=0.9
# initialize all others with 0.5 ("average")
data[:,4:]=0.5
for i in range(4,21):
    # now vary each parameter smoothly based on position and with the above␣
 ↪randomness
    B=np.random.random(3)-0.5
    data[:,i]+=.3*np.einsum(B,[0],dataPre,[1,0],[1])
```

```
# select one sample and 10 attributes at random and perturb the corresponding␣
 ↪values
i=np.random.randint(nPts*nPts)
j=np.random.choice(np.arange(4,21),replace=False,size=10)
nAttr=j.shape[0]
data[i,j]+=0.6*(np.random.random(size=nAttr)-0.5)
```

[29]:
```
# now plot all faces
fig=plt.figure(figsize=(7,7))
ax=fig.add_subplot()
for j,x in enumerate(data):
    cface(ax,*x)
    ax.text(x[0],x[1]-0.5/nPts,j)
plt.xlim([-sizeparam2/nPts,(nPts-1+sizeparam2)/nPts])
plt.ylim([-sizeparam2/nPts,(nPts-1+sizeparam2)/nPts])
plt.axis("off")
plt.show()
```



[30]:
```
i
```

[30]: 15

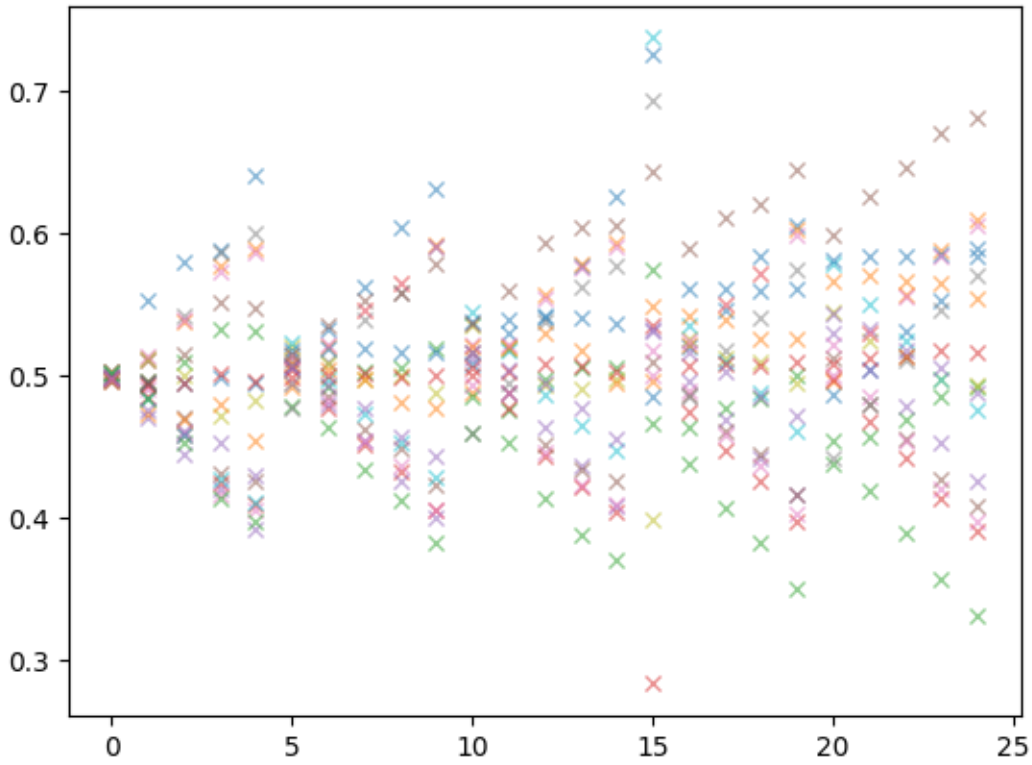
1.4 Alternative Visualizations

[31]:
```
data.shape
```

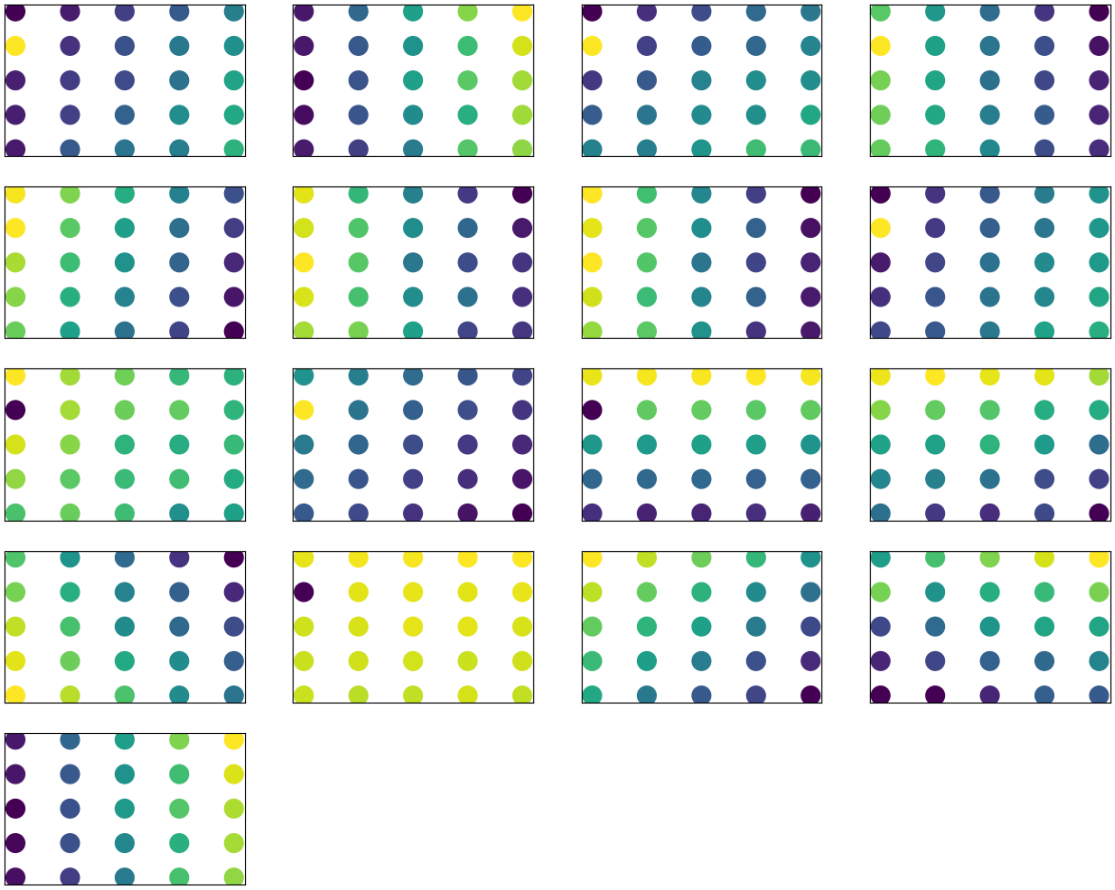
[31]: (25, 21)

[32]:
```
# primitive alternative visualization
# each x-value represents one sample, all values are simply shown as markers at␣
 ↪y-positions, with different colors
```

```
for i in range(4,21):
    plt.plot(data[:,i],alpha=0.5,marker="x",lw=0)
plt.show()
```



[33]:
```
# alternative visualization as many small scatter plots
# one attribute per scatter plot, one dot per sample (always at same position␣
  ↪as face), value encoded as color
nCols=5
jList=np.arange(4,21)
nRows=(len(jList)-1)//nCols+1

fig=plt.figure(figsize=(3*nCols,3*nRows))
for k,j in enumerate(jList):
    ax=fig.add_subplot(nCols,nRows,k+1)
    plt.scatter(data[:,0],data[:,1],c=data[:,j],s=200)
    #for i2,x in enumerate(data[:,:2]):
    #    ax.text(x[0],x[1]-0.5/nPts,i2)
    plt.xticks([])
    plt.yticks([])
plt.show()
```



1.5 Revisit same problem with PCA

[34]:
```
# do simple PCA on data matrix
# dataMat is assumed to be centered, matrix of shape (nSamples,dimSample)
def PCA(dataMat,keep=None):
    nSamples,dim=dataMat.shape
    if dim<nSamples:
        if keep is None:
            keep=dim
        A=dataMat.transpose().dot(dataMat)/nSamples
        eigData=np.linalg.eigh(A)
        eigval=(eigData[0][-keep::])[::-1]
        eigvec=((eigData[1][:,-keep::]).transpose())[::-1]
    else:
        if keep is None:
            keep=nSamples
        A=dataMat.dot(dataMat.transpose())/nSamples
        eigData=np.linalg.eigh(A)
```

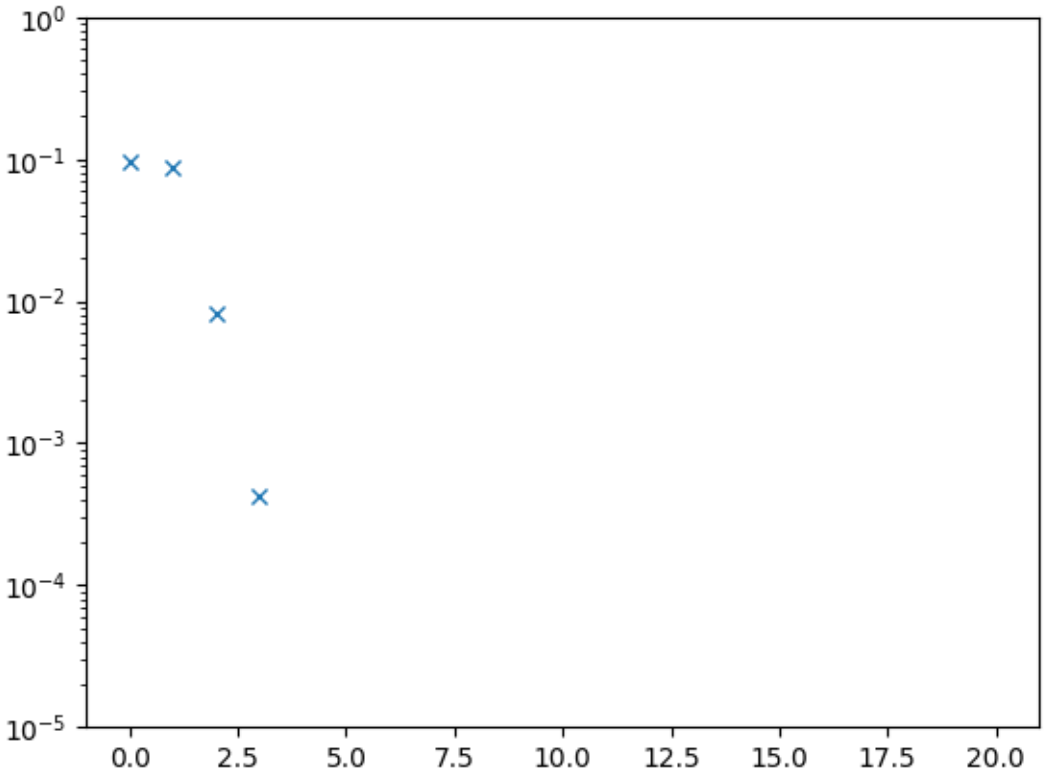
```
        eigval=(eigData[0][-keep::])[::-1]
        eigvec=((eigData[1][:,-keep::]).transpose())[::-1]

        eigvec=np.einsum(eigvec,[0,1],dataMat,[1,2],[0,2])
        # renormalize
        normList=np.linalg.norm(eigvec,axis=1)
        eigvec=np.einsum(eigvec,[0,1],1/normList,[0],[0,1])
    return eigval,eigvec
```

[35]:
```
data.shape
```

[35]: (25, 21)

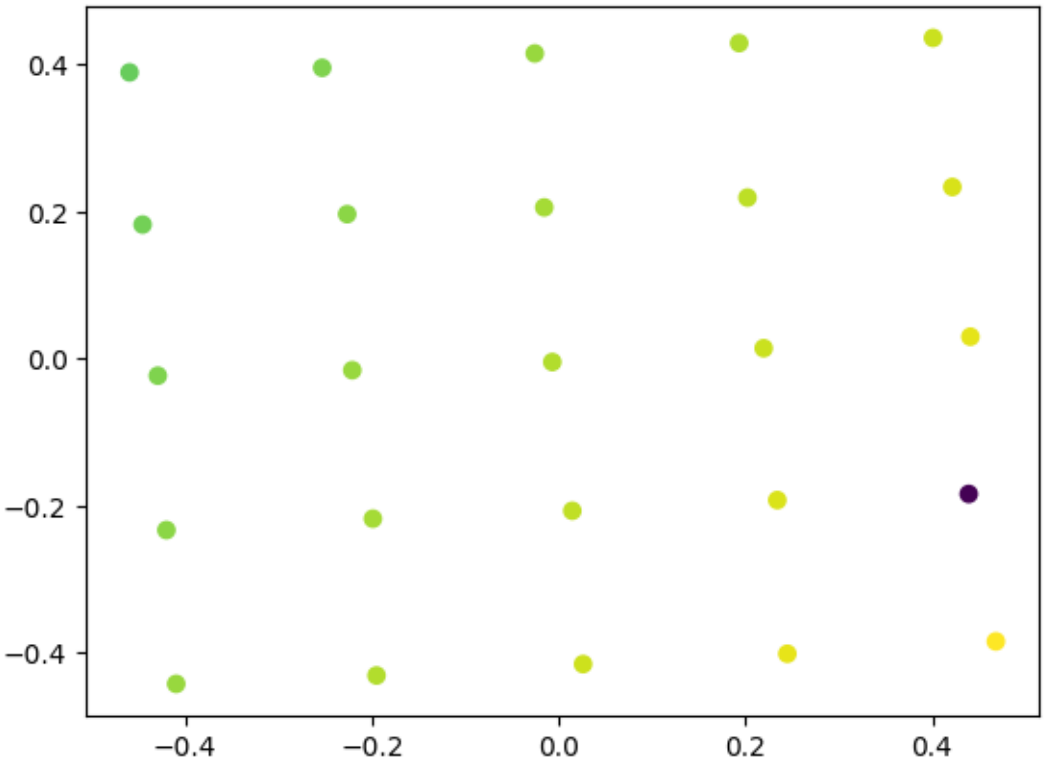
[36]:
```
dataNew=data.copy()
dataMean=np.mean(dataNew,axis=0)
dataNew-=dataMean
eigval,eigvec=PCA(dataNew)
```

[39]:
```
plt.plot(eigval,marker="x",lw=0)
plt.yscale("log")
plt.ylim([1E-5,1E0])
plt.show()
```



[40]:
```
coef=np.einsum(eigvec,[0,1],dataNew,[2,1],[2,0])
```

[41]:
```
plt.scatter(coef[:,0],coef[:,1],c=coef[:,2])
plt.show()
```



[ ]:

[ ]: