---------------------------------------导包-------------------------------------------

import scanpy as sc

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import math

from scipy import stats

-------------------------------------前置函数-------------------------------------------

def max\_min\_normalize2(list1): #减最小值进行归一化

list1=np.array(list1)

max\_value=list1.max()

min\_value=list1.min()

list2=((list1-min\_value)/(max\_value-min\_value))

return list2

def AnomalyDetection4(list1,list2): #二元独立高斯分布

list1=np.array(list1)

list2=np.array(list2)

list1\_mean=list1.mean()

list2\_mean=list2.mean()

list1\_std=list1.std()

list2\_std=list2.std()

list1\_p=(((2\*np.pi\*list1\_std\*\*2)\*\*0.5)\*\*(-1))\*np.exp(-1\*(list1-list1\_mean)\*\*2/(2\*list1\_std\*\*2))

list2\_p=(((2\*np.pi\*list2\_std\*\*2)\*\*0.5)\*\*(-1))\*np.exp(-1\*(list2-list2\_mean)\*\*2/(2\*list2\_std\*\*2))

list\_p=list1\_p\*list2\_p

list\_bool=(list\_p>=0.001)

list3=(list1[list\_bool])

list4=(list2[list\_bool])

return list3,list4

def delete\_the\_lowest\_possible\_point2(list1,number): #用numpy 做z\_score以3sigma原理删除边缘点

list1=np.array(list1)

list1\_mean=list1.mean()

list1\_std=list1.std()

index\_list=np.array(range(len(list1))).reshape(1,len(list1))

list1\_z\_score=abs((np.array(list1)-list1\_mean)/list1\_std).reshape(1,len(list1)) #把list转换成array直接进行加减乘除操作

list1\_p=np.concatenate((index\_list,list1\_z\_score),axis=0) #axis=1表示对应行的数组进行拼接,

list1\_p=list1\_p.T[np.lexsort(list1\_p)].T # 根据数组的第二行元素进行从小到大的排序

list2=list1\_p[0][:(len(list1\_p[0])-number)].astype(int) #用切片的方式，删除了低概率的number个点，再把点的序号返回给list2

list3=list1[list2]

return list3

def fit(x, y): #计算一元线性回归方程的a和b

x=np.array(x)

y=np.array(y)

x\_mean = x.mean()

y\_mean= y.mean()

tmp\_1=sum((x-x\_mean)\*(y-y\_mean))

tmp\_2=sum((x-x\_mean)\*\*2)

a = tmp\_1 / tmp\_2 # 计算a斜率的值

b = y\_mean - a \* x\_mean # 计算b截距的值

return a,b

#计算出一条经过最少点，覆盖下方所有点的上折线

def find\_the\_up\_line2(list1,list2):

list3=np.unique(list1)

list2\_highest=np.array([])

for i in list3:

value=max(list2[np.where(list1==i)[0]])

list2\_highest=np.append(list2\_highest,value)

list1=list3

list2=list2\_highest #排序结果输出正常

if len(list1)<=1 or len(list2)<=1:

return([],[])

up\_line1=[]

up\_line2=[]

mark\_point=0

up\_line1.append(list1[0])

up\_line2.append(list2[0])

while(mark\_point+1<len(list1)):

# print(mark\_point)

list3=list1.copy()[mark\_point+1:]

list4=list2.copy()[mark\_point+1:]

list\_k=(list4-list2[mark\_point])/(list3-list1[mark\_point])

list\_k\_mark\_point=np.where(list\_k==max(list\_k))[0][0] #list\_k中最大值的位置

mark\_point=mark\_point+list\_k\_mark\_point+1 #更新标记的位置

up\_line1.append(list1[mark\_point])

up\_line2.append(list2[mark\_point])

return up\_line1,up\_line2

#计算出一条经过最少点，兜住上方所有点的上折线

def find\_the\_under\_line2(list1,list2):

list3=np.unique(list1)

list2\_highest=np.array([])

for i in list3:

value=min(list2[np.where(list1==i)[0]])

list2\_highest=np.append(list2\_highest,value)

list1=list3

list2=list2\_highest #排序结果输出正常

if len(list1)<=1 or len(list2)<=1:

return([],[])

under\_line1=[]

under\_line2=[]

mark\_point=0

under\_line1.append(list1[0])

under\_line2.append(list2[0])

while(mark\_point+1<len(list1)):

list3=list1.copy()[mark\_point+1:]

list4=list2.copy()[mark\_point+1:]

list\_k=(list4-list2[mark\_point])/(list3-list1[mark\_point])

list\_k\_mark\_point=np.where(list\_k==min(list\_k))[0][0] #list\_k中最小值的位置

mark\_point=mark\_point+list\_k\_mark\_point+1 #更新标记的位置

under\_line1.append(list1[mark\_point])

under\_line2.append(list2[mark\_point])

return under\_line1,under\_line2

#为消除歧义性，将点到两条回归线L1和L2的距离相加，删除最终和值最大的一群点

def delete\_x\_point4(list1,list2,delete\_number):

list1=np.array(list1)

list2=np.array(list2)

a1,b1=fit(list1,list2) #为消除歧义性，用交集的方式筛选出距离两条回归线都远的点

a2,b2=fit(list2,list1)

Y1=list1\*a1+b1 #散点到线性回归线的距离

Y2=list2\*a2+b2 #交换x、y轴坐标后，散点到线性回归线的距离

dis\_list2\_to\_y1=np.array(abs(list2-Y1)).reshape(1,len(Y1))

dis\_list1\_to\_y2=np.array(abs(list1-Y2)).reshape(1,len(Y2))

index\_list=np.array(range(len(list1))).reshape(1,len(list1))

dis\_sum=dis\_list2\_to\_y1+dis\_list1\_to\_y2 #距离求和

distance3=np.concatenate((index\_list,dis\_sum),axis=0) #axis=1表示对应行的数组进行拼接,

distance3=distance3.T[np.lexsort(distance3)].T # 根据数组的第二行元素进行从小到大的排序

index\_list=distance3[0][:len(distance3[0])-delete\_number] #用切片的方式，删除数组后面的元素

list1=list1[index\_list.astype(int)]#distance3[0].astype(int)是未删除的散点序号

list2=list2[index\_list.astype(int)]

return(list1,list2)

#计算得上下折线中间的面积S1或S2，与最高最低y轴线、最左最右x轴线的围成面积S的比值

def caculate\_the\_Square(x1,y1,x2,y2):

square1=np.trapz(y1,x1)

square2=np.trapz(y2,x2)

Original\_Square=(max((x1),default=0)-min((x1),default=0))\*(max((y1),default=0)-min((y2),default=0))

Original\_Square\_radio=round((square1-square2)/Original\_Square,4)

return((square1-square2),Original\_Square\_radio,Original\_Square)

--------------------------------点类型划分函数，一起算了Dsa------------------------------------------------------

#在清理点的过程中，将散点分为三组Sc1、Sc2和Sc3，其中Sc1（近原点的点）被舍弃不做下一步使用，

#Sc2（一个坐标趋于0的点）用来计算Dsz，Sc3传播下去，做下一步分析。

#clean\_data12中，times\_near0\_to\_no00对应N，radio\_y\_near\_zero对应Tx，radio\_x\_near\_zero对应Ty

def clean\_data12(list1,list2):

#清理数据中存在Nan的数据点,清理数据中存在0的数据点,记录单项为0的点的数量

#根据单项为零的点的分布和数量，判断两个向量的相关性

dec\_sim\_near\_zero=0

list1=np.array(list1) #清理nan

list2=np.array(list2)

list1\_bool=np.isnan(list1)

list2\_bool=np.isnan(list2)

bool\_list=np.array((list1\_bool+list2\_bool)==0)

list1=list1[bool\_list]

list2=list2[bool\_list]

list1=np.around(list1, 2)

list2=np.around(list2, 2)

list1=max\_min\_normalize2(list1) #减最小值进行归一化，范围缩放到0~1

list2=max\_min\_normalize2(list2)

list1,list2=AnomalyDetection4(list1,list2)

lenght\_list1=len(list1) #剔除在含有Nan的维度之后，向量的长度

if list1.size!=0 and list2.size!=0:

list1\_max=list1.max() #这里提取最大最小值是探讨近零维度用的

list2\_max=list2.max()

list1\_min=list1.min()

list2\_min=list2.min()

else:

list1\_max=0 #这里提取最大最小值是探讨近零维度用的

list2\_max=0

list1\_min=0

list2\_min=0

near\_x=list2\_min+0.03\*(list2\_max-list2\_min) #立一个标准值，低于这个值视为x轴上的点

near\_y=list1\_min+0.03\*(list1\_max-list1\_min) #立一个标准值，低于这个值视为y轴上的点

list1\_bool=(list1==0 ) #清理两个都是0的

list2\_bool=(list2==0)

bool\_list=((list1\_bool.astype(int)+list2\_bool.astype(int))==2)==0

list1=list1[bool\_list]

list2=list2[bool\_list]

list1\_bool=list1<=near\_y #清理单维度近0的

list2\_bool=list2<=near\_x

list\_x\_near\_zero=list2[list1\_bool]

list\_y\_near\_zero=list1[list2\_bool]

bool\_list=(list1\_bool+list2\_bool)==0 #提取两个维度都不近0的

list1=list1[bool\_list]

list2=list2[bool\_list]

number\_no\_zero=len(list1) #在两个向量中都非零的维度的数量

count\_near0=len(list\_x\_near\_zero)+len(list\_y\_near\_zero)

if len(list1)!=0:

times\_near0\_to\_no0=count\_near0/len(list1)

else:

times\_near0\_to\_no0=13

if len(list1)!=0 and times\_near0\_to\_no0<13: #所以点都接近x轴，y轴，直接判断相关性为0

list1\_max=list1.max() #这里提取最大最小值是探讨近零维度的分布与不近零维度的分布占比用的

list2\_max=list2.max()

list1\_min=list1.min()

list2\_min=list2.min()

if times\_near0\_to\_no0>0.1 :

times\_near0\_to\_no00=int(count\_near0/len(list1))

radio\_x\_near\_zero=0

radio\_y\_near\_zero=0

if len(list\_y\_near\_zero)>1: #list\_y\_near\_zero向量，即落在x轴上的点（y=0）

if len(set(list\_y\_near\_zero))>1:

mark\_point=int(len(list\_y\_near\_zero)/20)+1 #少量下降点最大值的取值，希望能避开异常值。这里选择取大于95%的数

if mark\_point==0:

mark\_point+=1

list\_y\_near\_zero=delete\_the\_lowest\_possible\_point2(list\_y\_near\_zero.tolist(),mark\_point)

list\_y\_near\_zero=np.array(list\_y\_near\_zero)

list\_y\_near\_zero\_max=list\_y\_near\_zero.max()

list\_y\_near\_zero\_min=list\_y\_near\_zero.min()

if (list1\_max-list1\_min)==0:

radio\_y\_near\_zero=1

else:

radio\_y\_near\_zero=(list\_y\_near\_zero\_max-list\_y\_near\_zero\_min)/(list1\_max-list1\_min)

else:

radio\_y\_near\_zero=0

if len(list\_x\_near\_zero)>1: #list\_y\_single\_zero向量，即落在y轴上的点（x=0）

if len(set(list\_x\_near\_zero))>1:

mark\_point=int(len(list\_x\_near\_zero)/20)+1

if mark\_point==0:

mark\_point+=1

list\_x\_near\_zero=delete\_the\_lowest\_possible\_point2(list\_x\_near\_zero.tolist(),mark\_point)

list\_x\_near\_zero=np.array(list\_x\_near\_zero)

list\_x\_near\_zero\_max=list\_x\_near\_zero.max()

list\_x\_near\_zero\_min=list\_x\_near\_zero.min()

if (list2\_max-list2\_min)==0:

radio\_x\_near\_zero=1

else:

radio\_x\_near\_zero=(list\_x\_near\_zero\_max-list\_x\_near\_zero\_min)/(list2\_max-list2\_min)

else:

radio\_x\_near\_zero=0

if radio\_x\_near\_zero>radio\_y\_near\_zero:

radio\_x\_near\_zero=radio\_x\_near\_zero\*(1.5\*\*times\_near0\_to\_no00)

dec\_sim\_near\_zero=radio\_x\_near\_zero\*times\_near0\_to\_no0

else:

radio\_y\_near\_zero=radio\_y\_near\_zero\*(1.5\*\*times\_near0\_to\_no00)

dec\_sim\_near\_zero=radio\_y\_near\_zero\*times\_near0\_to\_no0

else:

dec\_sim\_near\_zero=0

radio\_near\_zero=0

else:

dec\_sim\_near\_zero=1

return(list1,list2,dec\_sim\_near\_zero,lenght\_list1,count\_near0,number\_no\_zero)

------------------------------------------计算Dsa的函数--------------------------------------------------

#计算θL1和θL2，并根据θL1和θL2计算Dsa，函数中dec\_sim对应Dsa,direction对应Dir

def caculate\_the\_regression\_and\_decrease\_sim5(list1,list2,R1,R2):

a1,b1=fit(list1,list2)

a2,b2=fit(list2,list1)

a1\_copy=a1 #在后面不知道哪里的机制把a更改了，所以这里做个小备份

a2\_copy=a2

b1\_copy=b1

b2\_copy=b2

A3=0

B3=0

list1=np.array(list1)

list2=np.array(list2)

sum\_list3=0 #计算点在拟合的线性回归线上聚集吗

sum\_list1=sum(abs(list1)) #计算点在X轴聚集吗

sum\_list2=sum(abs(list2)) #计算点在y轴聚集吗

if( abs(round(abs(math.atan(a1)/math.pi\*180),2)-45)>abs(round(abs(math.atan(a2)/math.pi\*180),2)-45)):

A3=a1\_copy

B3=b1\_copy

sum\_list3=sum(abs(list2-(A3\*list1+B3)))

else:

A3=a2\_copy

B3=b2\_copy

sum\_list3=sum(abs(list1-(A3\*list2+B3)))

if(round((math.atan(A3)/math.pi\*180),2)>=0):

direction=1

else:

direction=-1

if sum\_list3<sum\_list1 and sum\_list3<sum\_list2:

angle\_to\_45=round(abs(abs(math.atan(A3)/math.pi\*180)-45),2) #回归线与45度角的夹角

if R1<=0.50 and R2 <=0.40: #R1、R2较小的区别对待，因为矩形面积中，面积占比较小表示点分布成狭长状，目的是减小Pearson在0.7段时，diy方法的误判率

if angle\_to\_45>25: #15度开始加入dec\_sim的影响，

if angle\_to\_45>=35:

dec\_sim=round((angle\_to\_45/45)\*\*5,4)

else:

dec\_sim=round((angle\_to\_45/45)\*\*(10-((angle\_to\_45-25)\*0.5)),4)

else:

dec\_sim=0

else:

if angle\_to\_45>20:

dec\_sim=round((angle\_to\_45/45)\*\*3.5,4)

else:

dec\_sim=0

else:

dec\_sim=1

return dec\_sim,direction

---------------------------------------计算STgcor--------------------------------------------------------------------

#diy\_sim8的结果即是STgcor，其中dec\_sim为Dsa，dec\_sim\_by\_single0为Dsz

def diy\_sim8(list1,list2):

list1,list2,dec\_sim\_by\_single0,lenght\_list1,count\_single0,number\_no\_zero=clean\_data12(list1,list2) #清理含零的点，统计单零的点

if dec\_sim\_by\_single0>=1:

return 0

x1,y1=find\_the\_up\_line2(list1,list2)

x2,y2=find\_the\_under\_line2(list1,list2)

if len(x1)>=2:

if len(list1)<10: #根据点的数量，选定要删除的点的数量

delete\_number=1

else:

delete\_number=int(len(list1)\*0.1)

list1,list2=delete\_x\_point4(list1,list2,delete\_number) #调用删除点的函数，删除指定数量点

x1,y1=find\_the\_up\_line2(list1,list2)

x2,y2=find\_the\_under\_line2(list1,list2)

if len(x1)>=2:

Square1,R1,Original\_Square1=caculate\_the\_Square(x1,y1,x2,y2)

R1=round(Square1/Original\_Square1,4)

list1,list2=delete\_x\_point4(list1,list2,delete\_number)

x1,y1=find\_the\_up\_line2(list1,list2)

x2,y2=find\_the\_under\_line2(list1,list2)

Square2,R2,Original\_Square2=caculate\_the\_Square(x1,y1,x2,y2)

R2=round(Square2/Original\_Square1,4)

dec\_sim,direction=caculate\_the\_regression\_and\_decrease\_sim5(list1,list2,R1,R2)

if (math.isnan(R1)|math.isnan(R2)|math.isnan(dec\_sim)):

result=0

else:

result=1-round(R1\*\*(2.55-int(R1\*10)\*0.1),2)-round(R2\*\*(2.35-int(R2\*10)\*0.1),2)-round((R1-R2)\*\*(2-(R1-R2)\*3),4)-dec\_sim-round(dec\_sim\_by\_single0,2)

if result<0:

result=0

return result\*direction

else:

return 0

else:

return 0