|  |
| --- |
| 附录2 |
| 介绍： |
| 问题一数据预处理 import numpy as np import pandas as pd df = pd.read\_csv("../data/attachments2.csv",encoding='utf-8') df['销售日期'] = pd.to\_datetime(df['销售日期']) df = df.set\_index('销售日期') print(df)  #销售量按日期合并 df\_sum = pd.DataFrame([]) df\_sum['销售日期']=pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D') df\_sum = df\_sum.set\_index('销售日期') for index, row in df.iterrows():  #print(df.loc[index])  #print(df\_sum.loc[index])  name = str(row['单品编码'])  if name in df\_sum:  df\_sum.at[index, name] += row['销量(千克)']  else:  df\_sum[name] = 0.0  df\_sum.at[index, name] = row['销量(千克)'] print(df\_sum) df\_sum.to\_csv('../data/销售量按日期单品合并.csv',sep=',',encoding='utf\_8\_sig')  #销售额按日期合并 df\_sum = pd.DataFrame([]) df\_sum['销售日期']=pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D') df\_sum = df\_sum.set\_index('销售日期') for index, row in df.iterrows():  #print(df.loc[index])  #print(df\_sum.loc[index])  name = str(row['单品编码'])  if name in df\_sum:  df\_sum.at[index, name] += row['销量(千克)']\*row['销售单价(元/千克)']  else:  df\_sum[name] = 0.0  df\_sum.at[index, name] = row['销量(千克)']\*row['销售单价(元/千克)'] print(df\_sum) df\_sum.to\_csv('../data/销售额按日期单品合并.csv',sep=',',encoding='utf\_8\_sig')  Python 问题一Kmeans预测 import numpy as np import pandas as pd df = pd.read\_csv("../data/按日期单品合并.csv",encoding='utf-8',index\_col='销售日期') pl = pd.read\_excel("../data/附件1.xlsx") pl['单品编码'] = pl['单品编码'].astype(str) pl = pl.set\_index('单品编码') print(pl) # for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  temp = pd.DataFrame([])  for index in df:  plrow = pl.loc[index]  if plrow['分类名称'] == types:  name\_CN = plrow['单品名称']  temp[name\_CN] = df[index]  X = temp.T  from sklearn.cluster import KMeans  km = KMeans(n\_clusters=4,random\_state=666)  km.fit(X) # 无监督学习，拟合的时候不需要样本标签  y\_predict = km.predict(X) # 预测  from sklearn.metrics import calinski\_harabasz\_score  print(calinski\_harabasz\_score(X,y\_predict))  y\_predict = pd.DataFrame(y\_predict,X.T.columns)  y\_predict.rename(columns={'0':'聚类'})  #print(y\_predict)  y\_predict.to\_csv('../data/Kmeans/'+types+'Kmeans.csv',sep=',',encoding='utf\_8\_sig')   for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  temp = pd.DataFrame([])  for index in df:  plrow = pl.loc[index]  if plrow['分类名称'] == types:  name\_CN = plrow['单品名称']  temp[name\_CN] = df[index]  X = temp.T  from sklearn.cluster import KMeans  SEE = np.empty(shape=(X.shape[0]-1,1))  CH = np.empty(shape=(X.shape[0]-1,1))  for index in range(2,X.shape[0]-1):  km = KMeans(n\_clusters=index,random\_state=666)  km.fit(X) # 无监督学习，拟合的时候不需要样本标签  y\_predict = km.predict(X) # 预测  SEE[index] = km.inertia\_  from sklearn.metrics import calinski\_harabasz\_score  CH[index] = calinski\_harabasz\_score(X,y\_predict)  y\_predict = pd.DataFrame(y\_predict,X.T.columns)  y\_predict.rename(columns={'0':'聚类'})  #print(y\_predict)  y\_predict.to\_csv('../data/Kmeans/'+types+str(index)+'Kmeans.csv',sep=',',encoding='utf\_8\_sig')  import matplotlib.pyplot as plt  plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体  plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题  fig = plt.figure(figsize=(20,10))  plt.xticks(rotation=270, fontsize= 30)  plt.yticks(fontsize=20)  plt.title(types+'SSE随聚类数量变化的折线图', fontsize= 30)  plt.plot(SEE, linewidth = 3.0)  plt.savefig('../data/Kmeans/LineChart/'+types+'SSE随聚类数量变化的折线图.jpg')  plt.show()   plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体  plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题  fig = plt.figure(figsize=(20,10))  plt.xticks(rotation=270, fontsize= 30)  plt.yticks(fontsize=20)  plt.title(types+'CH随聚类数量变化的折线图', fontsize= 30)  plt.plot(CH, linewidth = 3.0)  plt.savefig('../data/Kmeans/LineChart/'+types+'CH随聚类数量变化的折线图.jpg')  plt.show()  #print(SEE)  #print(CH)  Python 问题一Pearson相关性 import numpy as np import pandas as pd df = pd.read\_csv("../data/按日期单品合并.csv",encoding='utf-8',index\_col='销售日期') pl = pd.read\_excel("../data/附件1.xlsx")  pl['单品编码'] = pl['单品编码'].astype(str) pl = pl.set\_index('单品编码') print(pl)  for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  temp = pd.DataFrame([])  for index in df:  plrow = pl.loc[index]  if plrow['分类名称'] == types:  name\_CN = plrow['单品名称']  temp[name\_CN] = df[index]  corr = temp.corr(method='pearson')  corr = np.absolute(corr)  corr.to\_csv('../data/corr/'+types+'corr.csv',sep=',',encoding='utf\_8\_sig')   from matplotlib import cm  import matplotlib.pyplot as plt  plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体  plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题  cmap = cm.Blues  fig = plt.figure(figsize=(30,30))  ax=fig.add\_subplot(111)  map = ax.imshow(corr, interpolation='nearest', cmap=cmap, vmin=0, vmax=1)  plt.title(types+'correlation coefficient--headmap')  ax.set\_yticks(range(len(corr.columns)))  ax.set\_yticklabels(corr.columns)  ax.set\_xticks(range(len(corr)))  ax.set\_xticklabels(corr.columns)  plt.colorbar(map)  plt.savefig('../data/corr/'+types+'corr.jpg')  plt.show()  Python 问题一折线图绘制 import numpy as np import pandas as pd df = pd.read\_csv("../data/按日期单品合并.csv",encoding='utf-8',index\_col='销售日期') pl = pd.read\_excel("../data/附件1.xlsx") pl['单品编码'] = pl['单品编码'].astype(str) pl = pl.set\_index('单品编码') print(pl)  for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  names = pd.read\_excel("../data/Kmeans/representative/"+types+'.xlsx', header=None)  temp = pd.DataFrame([])  for index in df:  plrow = pl.loc[index]  if plrow['分类名称'] == types:  name\_CN = plrow['单品名称']  temp[name\_CN] = df[index]  temp['Sum'] = temp.apply(lambda row: row.sum(), axis=1)  #print(temp)  import matplotlib.pyplot as plt  plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体  plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题  fig= plt.figure(figsize=(30,15))  plt.xticks(rotation=270, fontsize= 10)  import matplotlib.dates as mdates  ax=plt.gca()  #ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m'))  ax.xaxis.set\_major\_locator(mdates.MonthLocator(interval=2))  plt.title('代理单品占'+types+'下的销售量百分比随着时间变化而变化的折线',fontsize = 20)  handles = []  for index in names[0]:  Category\_proportion = temp[index]/temp['Sum']  #print(Category\_proportion)  plt.plot(Category\_proportion, label=index)  plt.legend() # 生成图例  plt.savefig('../data/linecharts/Types/'+'代理单品占'+types+'下的销售量百分比随着时间变化而变化的折线.jpg')  plt.show()  Python 问题一主成分分析 import numpy as np import pandas as pd df = pd.read\_csv("../data/按日期单品合并.csv",encoding='utf-8',index\_col='销售日期') pl = pd.read\_excel("../data/附件1.xlsx") pl['单品编码'] = pl['单品编码'].astype(str) pl = pl.set\_index('单品名称') print(pl)  df\_sum = pd.DataFrame([]) df\_sum['销售日期']=pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D') df\_sum = df\_sum.set\_index('销售日期') RANK = pd.DataFrame(data = [11,3,4,6,6,10], index = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌'] , columns=['Rank']) for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  names = pd.read\_excel("../data/Kmeans/representative/"+types+'.xlsx', header=None)  temp = pd.DataFrame([])  for name in names[0]:  number = pl.loc[name]['单品编码']  temp[name] = df[number]  #print(temp)  # 对每一列标准化  # 创建StandardScaler对象  from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  # 对每一列进行标准化  temp\_standardized = pd.DataFrame(scaler.fit\_transform(temp), columns=temp.columns)   from sklearn.decomposition import PCA  pca = PCA(n\_components=RANK.loc[types]['Rank'])  pca.fit(temp\_standardized)  pca\_matric = pca.transform(temp\_standardized)  # 转置  pca\_matric\_T = pca\_matric.T  print('-'\*20)  print(types)  print(pca.explained\_variance\_ratio\_)  weight = pca.explained\_variance\_ratio\_  X = np.array(temp\_standardized)  COV = temp\_standardized.cov()/1.000914  #print(COV)  pd.DataFrame(COV).to\_csv('../data/Kmeans/'+types+'COV.csv',sep=',',encoding='utf\_8\_sig')   W, V = np.linalg.eig(COV)  print("特征值")  print(W)  print("特征向量")  ww = []  for i in range(RANK.loc[types]['Rank']):  print(i)  print(V.T[i])  print(V.T[i]\*weight[i])  temp = V.T[i]\*weight[i]  ww.append(temp.tolist())   w\_sum = []  weight\_sum = weight.sum()  print("权值相加为：")  print(weight\_sum)  print("向量为：")  for i in range(temp.shape[0]):  w\_sum.append(np.array(ww).T[i].sum()/weight\_sum)  print(w\_sum)  print('归一化后为')  w\_sum\_nor = w\_sum/np.array(w\_sum).sum()  print(w\_sum\_nor)   m,n = np.array(temp\_standardized).shape  emotion = []  matrix\_np = np.array(temp\_standardized)  for i in range(m):  # 加权降维  #print(matrix\_np\_T[i])  emotion.append((matrix\_np[i]\*w\_sum\_nor).sum())  df\_sum[types] = emotion  print(df\_sum) df\_sum.to\_csv('../data/Kmeans/representative/'+'代理指数.csv',sep=',',encoding='utf\_8\_sig')  corr = df\_sum.corr(method='pearson') corr = np.absolute(corr) corr.to\_csv('../data/corr/'+'AllTypesCorr.csv',sep=',',encoding='utf\_8\_sig') from matplotlib import cm import matplotlib.pyplot as plt plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体 plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题 cmap = cm.Blues fig = plt.figure(figsize=(30,30)) ax=fig.add\_subplot(111) map = ax.imshow(corr, interpolation='nearest', cmap=cmap, vmin=0, vmax=1) plt.title('All Types Correlation coefficient Heatmap', fontsize= 50) ax.set\_yticks(range(len(corr.columns))) ax.set\_yticklabels(corr.columns) ax.set\_xticks(range(len(corr))) ax.set\_xticklabels(corr.columns) plt.xticks(rotation=270, fontsize= 30) plt.yticks(fontsize= 30) plt.colorbar(map) plt.savefig('../data/corr/'+'AllTypesCorr.jpg') plt.show()  Python 问题一饼状图 import numpy as np import pandas as pd df = pd.read\_csv("../data/按日期单品合并.csv",encoding='utf-8',index\_col='销售日期') pl = pd.read\_excel("../data/附件1.xlsx") pl['单品编码'] = pl['单品编码'].astype(str) pl = pl.set\_index('单品编码') print(pl)  sum\_per\_column = pd.DataFrame(df.sum()).T print(sum\_per\_column)   for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  temp = pd.DataFrame([])  for index in sum\_per\_column:  plrow = pl.loc[index]  if plrow['分类名称'] == types:  name\_CN = plrow['单品名称']  temp[name\_CN] = sum\_per\_column[index]   import matplotlib.pyplot as plt  from matplotlib import cm  plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体  plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题  fig = plt.figure(figsize=(60,30))  # 数据  labels = temp.columns  sizes = temp.values[0]  colors = cm.rainbow(np.arange(len(sizes))/len(sizes)) # colormaps: Paired, autumn, rainbow, gray,spring,Darks patches, texts, autotexts = ax.pie(sizes, labels=labels, autopct='%1.0f%%',shadow=False, startangle=170, colors=colors)  # 绘制饼图  plt.pie(sizes, labels=labels, colors = colors, autopct='%1.1f%%')  plt.legend(fontsize = 10 , loc = 'best')  # 设置图表标题  plt.title(types+'各单品销售量占比图',fontsize = 40)  plt.savefig('../data/piecharts/'+types+'PieChartNew.jpg')  # 显示图表  #plt.show()  print(temp.T)  temp.T.to\_csv('../data/piecharts/'+types+'.csv', sep= ',', encoding='utf\_8\_sig')  Python 问题一品类折线图绘制 import numpy as np import pandas as pd df = pd.read\_csv("../data/按日期单品合并.csv",encoding='utf-8',index\_col='销售日期') pl = pd.read\_excel("../data/附件1.xlsx") pl['单品编码'] = pl['单品编码'].astype(str) pl = pl.set\_index('单品编码') print(pl) df['Sum'] = df.apply(lambda row: row.sum(), axis=1)  df\_sum = [] df\_types\_sum = pd.DataFrame([]) df\_types\_sum['销售日期']=pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D') df\_types\_sum = df\_types\_sum.set\_index('销售日期') import matplotlib.pyplot as plt plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体 plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题 fig= plt.figure(figsize=(30,15)) plt.xticks(rotation=270, fontsize= 10) import matplotlib.dates as mdates ax=plt.gca() #ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m')) ax.xaxis.set\_major\_locator(mdates.MonthLocator(interval=2)) plt.title('各品类销售量百分比随着时间变化而变化的折线',fontsize = 20) for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  names = pd.read\_excel("../data/Kmeans/representative/"+types+'.xlsx', header=None)  temp = pd.DataFrame([])  for index in df:  if index == 'Sum':  break;  plrow = pl.loc[index]  if plrow['分类名称'] == types:  name\_CN = plrow['单品名称']  temp[name\_CN] = df[index]  temp['Sum'] = temp.apply(lambda row: row.sum(), axis=1)  #print(temp)  df\_sum.append(temp['Sum'].sum())  df\_types\_sum[types] = temp['Sum'].values  Category\_proportion = temp['Sum']/df['Sum']  #print(Category\_proportion)  plt.plot(Category\_proportion, label=types) plt.legend() # 生成图例 plt.savefig('../data/linecharts/TypesInAll/'+'各品类销售量百分比随着时间变化而变化的折线.jpg') plt.show()  print(df\_sum) df\_types\_sum.to\_csv('../data/销售量按日期品类合并.csv',sep=',',encoding='utf\_8\_sig')  df\_sum = pd.DataFrame(data = df\_sum, index = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌'] , columns=['Sum']) print(df\_sum) import matplotlib.pyplot as plt from matplotlib import cm plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体 plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题 fig = plt.figure(figsize=(5,5)) # 数据 labels = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌'] sizes = df\_sum['Sum'] colors = cm.rainbow(np.arange(len(sizes))/len(sizes)) # colormaps: Paired, autumn, rainbow, gray,spring,Darks patches, texts, autotexts = ax.pie(sizes, labels=labels, autopct='%1.0f%%',shadow=False, startangle=170, colors=colors) # 绘制饼图 plt.pie(sizes, labels=labels, colors = colors, autopct='%1.1f%%') # 设置图表标题 plt.title('各品类销售量饼图') plt.savefig('../data/piecharts/'+'各品类销售量饼图.jpg') # 显示图表 plt.show()  Python 问题二Alpha折线图绘制 import numpy as np import pandas as pd costPrice = pd.read\_csv("../data/成本价按日期单品合并.csv",encoding='utf-8') print(costPrice) price = pd.read\_csv("../data/单价按日期单品合并.csv",encoding='utf-8') print(price)  alpha = pd.DataFrame([]) alpha['销售日期']=pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D') alpha = alpha.set\_index('销售日期')   name = '102900005116714'  for index in price:  for index1 in costPrice:  if index[:-1] == name and index1 == name:  #print(price[index])  #print(costPrice[index1])  alpha['alpha'] = (price[index].values/costPrice[index1].values - 1)\*100  alpha = alpha.mask(alpha > 1000, 0) print(alpha)  import matplotlib.pyplot as plt plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体 plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题 fig= plt.figure(figsize=(30,15)) plt.xticks(rotation=270, fontsize= 10) import matplotlib.dates as mdates ax=plt.gca() ax.xaxis.set\_major\_locator(mdates.MonthLocator(interval=2)) plt.title('Alpha随着时间变化而变化的折线',fontsize = 20) plt.plot(alpha, label='Alpha') plt.legend() # 生成图例 plt.savefig('../data/linecharts/Alpha随着时间变化而变化的折线.jpg') plt.show()  Python 问题二BasinHopping import numpy as np from scipy.optimize import dual\_annealing,basinhopping,differential\_evolution m = 0.0975 def func2d(Q,D,D1,Q1,P):  f = np.min(Q,D+np.max(0,D1,Q1)\*(1-m))\*P  df = np.zeros(2)  return f,df  minimizer\_kwargs = {"method":"L-BFGS-B","jac":True} x0 = [1.0,1.0] ret = basinhopping(func2d,x0,minimizer\_kwargs=minimizer\_kwargs,niter=200) print("Global minimum:x = [%.4f,%.4f],f(x0) = %.4f" %(ret.x[0],ret.x[1],ret.fun))  Python 问题二 代理指数映射到销售量 import numpy as np import pandas as pd emotion = pd.read\_csv("../data/各品类每日代理指数.csv",encoding='utf-8',index\_col='销售日期') emotion\_pre = pd.read\_csv("../data/代理指数未来一周预测.csv",encoding='utf-8',index\_col='销售日期') volume = pd.read\_csv("../data/销售量按日期品类合并.csv",encoding='utf-8',index\_col='销售日期')  print(emotion) print(emotion\_pre) print(volume)  emotion\_merged = pd.concat([emotion,emotion\_pre]) print(emotion\_merged)  from sklearn.preprocessing import MinMaxScaler emotion\_ans = pd.DataFrame([]) emotion\_ans['销售日期']=pd.date\_range(start='2020-07-01', end='2023-07-07', freq='D') emotion\_ans = emotion\_ans.set\_index('销售日期') volume\_ans = pd.DataFrame([]) volume\_ans['销售日期']=pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D') volume\_ans = volume\_ans.set\_index('销售日期') for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  scaler = MinMaxScaler(feature\_range=(0, 1))  emotion\_temp = pd.DataFrame([])  emotion\_temp['销售日期']=pd.date\_range(start='2020-07-01', end='2023-07-07', freq='D')  emotion\_temp = emotion\_temp.set\_index('销售日期')  emotion\_temp[types] = emotion\_merged[types].values  emotion\_temp = scaler.fit\_transform(emotion\_temp)  #print(emotion\_temp)  emotion\_ans[types] = emotion\_temp   volume\_temp = pd.DataFrame([])  volume\_temp['销售日期'] = pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D')  volume\_temp = volume\_temp.set\_index('销售日期')  volume\_temp[types] = volume[types].values  scaler1 = MinMaxScaler(feature\_range=(0, 1))  volume\_temp = scaler1.fit\_transform(volume\_temp)  #print(volume\_temp)  volume\_ans[types] = volume\_temp   emotion\_temp = scaler1.inverse\_transform(emotion\_temp)   print(emotion\_temp) emotion\_ans.to\_csv('../data/emotion/情绪指数01映射.csv',sep=',',encoding='utf\_8\_sig') volume\_ans.to\_csv('../data/emotion/销售量01映射.csv',sep=',',encoding='utf\_8\_sig')  Python 问题二格兰杰因果检验 import numpy as np import pandas as pd emotion = pd.read\_csv("../data/各品类每日代理指数.csv",encoding='utf-8',index\_col='销售日期') price = pd.read\_csv("../data/各品类每日单价.csv",encoding='utf-8',index\_col='销售日期')  from statsmodels.tsa.stattools import grangercausalitytests for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  print('-'\*40)  print(types+': 代理指数 with 单价')  temp = pd.DataFrame([])  temp['emotion'] = emotion[types]  temp['price'] = price[types].values  grangercausalitytests(temp[['emotion','price']], maxlag=8)  print('-'\*40)  print(types+': 单价 with 代理指数')  grangercausalitytests(temp[['price','emotion']], maxlag=8)  volume = pd.read\_csv("../data/销售量按日期品类合并.csv",encoding='utf-8',index\_col='销售日期')  from statsmodels.tsa.stattools import grangercausalitytests for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  print('-'\*40)  print(types+': 销售量 with 单价')  temp = pd.DataFrame([])  temp['volume'] = volume[types]  temp['price'] = price[types].values  grangercausalitytests(temp[['volume','price']], maxlag=8)  print('-'\*40)  print(types+': 单价 with 销售量')  grangercausalitytests(temp[['price','volume']], maxlag=8)  Python 问题二各品类每日单价 import numpy as np import pandas as pd price = pd.read\_csv("../data/销售额按日期单品合并.csv",encoding='utf-8',index\_col='销售日期') volume = pd.read\_csv("../data/销售量按日期单品合并.csv",encoding='utf-8',index\_col='销售日期') pl = pd.read\_excel("../data/附件1.xlsx") pl['单品编码'] = pl['单品编码'].astype(str) pl = pl.set\_index('单品编码')  print(price) print(volume)  types\_per\_price = pd.DataFrame([]) for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  type\_price = pd.DataFrame([])  type\_volume = pd.DataFrame([])  for index in price:  plrow = pl.loc[index]  if plrow['分类名称'] == types:  name\_CN = plrow['单品名称']  type\_price[name\_CN] = price[index]  type\_volume[name\_CN] = volume[index]  #print(type\_price)  #print(type\_volume)  type\_price['Sum'] = type\_price.apply(lambda row: row.sum(), axis=1)  type\_volume['Sum'] = type\_volume.apply(lambda row: row.sum(), axis=1)  type\_per\_price = type\_price['Sum']/type\_volume['Sum']  types\_per\_price[types] = type\_per\_price  #print(type\_per\_price)  print(types\_per\_price) types\_per\_price.to\_csv('../data/各品类每日单价.csv',sep=',',encoding='utf\_8\_sig')  Python 问题二 向量自回归 import numpy as np import pandas as pd emotion = pd.read\_csv("../data/各品类每日代理指数.csv",encoding='utf-8',index\_col='销售日期') price = pd.read\_csv("../data/各品类每日单价.csv",encoding='utf-8',index\_col='销售日期')  LAG = pd.DataFrame(data = [1,1,5,1,3,5], index = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌'] , columns=['lag']) from statsmodels.tsa.api import VAR for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  maxlags = LAG.loc[types]['lag']  VAR\_data = []  VAR\_data.append(emotion[types].values[:-maxlags])  VAR\_data.append(price[types].values[maxlags:])  VAR\_data = pd.DataFrame(VAR\_data).T  #print(VAR\_data)  model = VAR(VAR\_data)   results = model.fit(maxlags=maxlags, ic='aic')  results.summary()    import matplotlib.pyplot as plt  import warnings  warnings.simplefilter(action='ignore', category=FutureWarning)  lag\_order = results.k\_ar  y1 = []  y2 = []  count = 0  n,m = VAR\_data.shape  # 用i枚举每一天  for i in range(n-maxlags-1):  # 制作输入  forecast\_input = VAR\_data.values[i:i+maxlags]  # 往前预测一天  forecast = results.forecast(forecast\_input, steps=1)  # 将预测结果加入数组中  y1.append(forecast[0][1])  # y2中存储的原本的值  y2.append(VAR\_data.values[i+maxlags+1][1])  # 将两者和阈值进行比较  if abs(forecast[0][1]-VAR\_data.values[i+maxlags+1][1]) < 0.17\*1e9:  count +=1   # 输出正确率  #print(count/(n-maxlags))  plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体  plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题  fig = plt.figure(figsize=(40,20))  # 手动对齐  plt.title(types+'每日单价预测',fontsize = 50)  plt.plot(y2[:-2],'r', label = '实际')  plt.plot(y1[2:],label = '预测')  plt.legend(fontsize = 50)  plt.savefig('../data/preLineCharts/'+types+'每日单价预测'+'.jpg')  plt.show()  LAG = pd.DataFrame(data = [1,5,5,3,1,5], index = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌'] , columns=['lag']) from statsmodels.tsa.api import VAR for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  maxlags = LAG.loc[types]['lag']  VAR\_data = []  VAR\_data.append(price[types].values[:-maxlags])  VAR\_data.append(emotion[types].values[maxlags:])  VAR\_data = pd.DataFrame(VAR\_data).T  #print(VAR\_data)  model = VAR(VAR\_data)   results = model.fit(maxlags=maxlags, ic='aic')  results.summary()    import matplotlib.pyplot as plt  import warnings  warnings.simplefilter(action='ignore', category=FutureWarning)  lag\_order = results.k\_ar  y1 = []  y2 = []  count = 0  n,m = VAR\_data.shape  # 用i枚举每一天  for i in range(n-maxlags-1):  # 制作输入  forecast\_input = VAR\_data.values[i:i+maxlags]  # 往前预测一天  forecast = results.forecast(forecast\_input, steps=1)  # 将预测结果加入数组中  y1.append(forecast[0][1])  # y2中存储的原本的值  y2.append(VAR\_data.values[i+maxlags+1][1])  # 将两者和阈值进行比较  if abs(forecast[0][1]-VAR\_data.values[i+maxlags+1][1]) < 0.17\*1e9:  count +=1   # 输出正确率  #print(count/(n-maxlags))  plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体  plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题  fig = plt.figure(figsize=(40,20))  # 手动对齐  plt.title(types+'每日代理指数预测',fontsize = 50)  plt.plot(y2[:-2],'r', label = '实际')  plt.plot(y1[2:],label = '预测')  plt.legend(fontsize = 50)  plt.savefig('../data/preLineCharts/'+types+'每日代理指数预测'+'.jpg')  plt.show()  LAG = pd.DataFrame(data = [1,1,5,1,3,5], index = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌'] , columns=['lag']) price\_pre = pd.DataFrame([]) price\_pre['销售日期']=pd.date\_range(start='2023-07-01', end='2023-07-07', freq='D') price\_pre = price\_pre.set\_index('销售日期') from statsmodels.tsa.api import VAR for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  maxlags = LAG.loc[types]['lag']  VAR\_data = []  VAR\_data.append(emotion[types].values[:-maxlags])  VAR\_data.append(price[types].values[maxlags:])  VAR\_data = pd.DataFrame(VAR\_data).T  #print(VAR\_data)  model = VAR(VAR\_data)   results = model.fit(maxlags=maxlags, ic='aic')  results.summary()   import warnings  warnings.simplefilter(action='ignore', category=FutureWarning)  lag\_order = results.k\_ar   count = 0  n,m = VAR\_data.shape  # 制作输入  forecast\_input = VAR\_data.values[-maxlags-1:]  #print(forecast\_input)  # 往前预测7天  forecast = results.forecast(forecast\_input, steps=7)  forecast = pd.DataFrame(forecast)[1]  #print(forecast)  price\_pre[types] = forecast.values print(price\_pre) price\_pre.to\_csv('../data/单价未来一周预测.csv',sep=',',encoding='utf\_8\_sig')  LAG = pd.DataFrame(data = [1,5,5,3,1,5], index = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌'] , columns=['lag']) emotion\_pre = pd.DataFrame([]) emotion\_pre['销售日期']=pd.date\_range(start='2023-07-01', end='2023-07-07', freq='D') emotion\_pre = emotion\_pre.set\_index('销售日期') from statsmodels.tsa.api import VAR for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  maxlags = LAG.loc[types]['lag']  VAR\_data = []  VAR\_data.append(price[types].values[:-maxlags])  VAR\_data.append(emotion[types].values[maxlags:])  VAR\_data = pd.DataFrame(VAR\_data).T  #print(VAR\_data)  model = VAR(VAR\_data)   results = model.fit(maxlags=maxlags, ic='aic')  results.summary()   import warnings  warnings.simplefilter(action='ignore', category=FutureWarning)  lag\_order = results.k\_ar   count = 0  n,m = VAR\_data.shape  # 制作输入  forecast\_input = VAR\_data.values[-maxlags-1:]  #print(forecast\_input)  # 往前预测7天  forecast = results.forecast(forecast\_input, steps=7)  forecast = pd.DataFrame(forecast)[1]  #print(forecast)  emotion\_pre[types] = forecast.values print(emotion\_pre) emotion\_pre.to\_csv('../data/代理指数未来一周预测.csv',sep=',',encoding='utf\_8\_sig')  Python 问题二 销售量向量自回归 import numpy as np import pandas as pd volume = pd.read\_csv("../data/销售量按日期品类合并.csv",encoding='utf-8',index\_col='销售日期') price = pd.read\_csv("../data/各品类每日单价.csv",encoding='utf-8',index\_col='销售日期')  LAG = pd.DataFrame(data = [1,1,1,1,1,1], index = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌'] , columns=['lag']) from statsmodels.tsa.api import VAR for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  maxlags = LAG.loc[types]['lag']  VAR\_data = []  VAR\_data.append(price[types].values[:-maxlags])  VAR\_data.append(volume[types].values[maxlags:])  VAR\_data = pd.DataFrame(VAR\_data).T  #print(VAR\_data)  model = VAR(VAR\_data)   results = model.fit(maxlags=maxlags, ic='aic')  results.summary()    import matplotlib.pyplot as plt  import warnings  warnings.simplefilter(action='ignore', category=FutureWarning)  lag\_order = results.k\_ar  y1 = []  y2 = []  count = 0  n,m = VAR\_data.shape  # 用i枚举每一天  for i in range(n-maxlags-1):  # 制作输入  forecast\_input = VAR\_data.values[i:i+maxlags]  # 往前预测一天  forecast = results.forecast(forecast\_input, steps=1)  # 将预测结果加入数组中  y1.append(forecast[0][1])  # y2中存储的原本的值  y2.append(VAR\_data.values[i+maxlags+1][1])  # 将两者和阈值进行比较  if abs(forecast[0][1]-VAR\_data.values[i+maxlags+1][1]) < 0.17\*1e9:  count +=1   # 输出正确率  #print(count/(n-maxlags))  plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体  plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题  fig = plt.figure(figsize=(40,20))  # 手动对齐  plt.title(types+'每日销售量预测',fontsize = 50)  plt.plot(y2[:-2],'r', label = '实际')  plt.plot(y1[2:],label = '预测')  plt.legend(fontsize = 50)  plt.savefig('../data/preLineCharts/'+types+'每日销售量预测'+'.jpg')  plt.show()  LAG = pd.DataFrame(data = [1,1,1,1,1,1], index = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌'] , columns=['lag']) volume\_pre = pd.DataFrame([]) volume\_pre['销售日期']=pd.date\_range(start='2023-07-01', end='2023-07-07', freq='D') volume\_pre = volume\_pre.set\_index('销售日期') from statsmodels.tsa.api import VAR for types in ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']:  maxlags = LAG.loc[types]['lag']  VAR\_data = []  VAR\_data.append(price[types].values[:-maxlags])  VAR\_data.append(volume[types].values[maxlags:])  VAR\_data = pd.DataFrame(VAR\_data).T  #print(VAR\_data)  model = VAR(VAR\_data)   results = model.fit(maxlags=maxlags, ic='aic')  results.summary()   import warnings  warnings.simplefilter(action='ignore', category=FutureWarning)  lag\_order = results.k\_ar   count = 0  n,m = VAR\_data.shape  # 制作输入  forecast\_input = VAR\_data.values[-maxlags-1:]  #print(forecast\_input)  # 往前预测7天  forecast = results.forecast(forecast\_input, steps=7)  forecast = pd.DataFrame(forecast)[1]  #print(forecast)  volume\_pre[types] = forecast.values print(volume\_pre) volume\_pre.to\_csv('../data/销售量未来一周预测.csv',sep=',',encoding='utf\_8\_sig')  Python 问题三 销售量LSTM import numpy as np import pandas as pd df = pd.read\_csv("../data/销售量按日期单品合并\_剔除后.csv",encoding='utf-8') print(df)  pl = pd.read\_excel("../data/附件1.xlsx") pl['单品编码'] = pl['单品编码'].astype(str) pl = pl.set\_index('单品编码')  # LSTM for international airline passengers problem with regression framing import numpy as np import matplotlib.pyplot as plt from pandas import read\_csv import math import tensorflow as tf from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense from tensorflow.keras.layers import LSTM from sklearn.preprocessing import MinMaxScaler from sklearn.metrics import mean\_squared\_error # convert an array of values into a dataset matrix def create\_dataset(dataset, look\_back=1):  dataX, dataY = [], []  for i in range(len(dataset)-look\_back-1):  a = dataset[i:(i+look\_back), 0]  dataX.append(a)  dataY.append(dataset[i + look\_back, 0])  return np.array(dataX), np.array(dataY) # fix random seed for reproducibility tf.random.set\_seed(7) # load the dataset dataframe = df[['销售日期','102900005115823 ']] dataframe.set\_index(["销售日期"], inplace=True) dataset = dataframe.values dataset = dataset.astype('float32') # normalize the dataset scaler = MinMaxScaler(feature\_range=(0, 1)) dataset = scaler.fit\_transform(dataset) # split into train and test sets train\_size = int(len(dataset) \* 0.67) test\_size = len(dataset) - train\_size train, test = dataset[0:train\_size,:], dataset[train\_size:len(dataset),:] # reshape into X=t and Y=t+1 look\_back = 7 trainX, trainY = create\_dataset(train, look\_back) testX, testY = create\_dataset(test, look\_back) # reshape input to be [samples, time steps, features] trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1])) testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1])) # create and fit the LSTM network model = Sequential() model.add(LSTM(30, input\_shape=(1, look\_back))) model.add(Dense(1)) model.compile(loss='mean\_squared\_error', optimizer='adam') model.fit(trainX, trainY, epochs=30, batch\_size=1, verbose=2) # make predictions trainPredict = model.predict(trainX) testPredict = model.predict(testX) # invert predictions trainPredict = scaler.inverse\_transform(trainPredict) trainY = scaler.inverse\_transform([trainY]) testPredict = scaler.inverse\_transform(testPredict) testY = scaler.inverse\_transform([testY]) # calculate root mean squared error trainScore = np.sqrt(mean\_squared\_error(trainY[0], trainPredict[:,0])) print('Train Score: %.2f RMSE' % (trainScore)) testScore = np.sqrt(mean\_squared\_error(testY[0], testPredict[:,0])) print('Test Score: %.2f RMSE' % (testScore)) # shift train predictions for plotting trainPredictPlot = np.empty\_like(dataset) trainPredictPlot[:, :] = np.nan trainPredictPlot[look\_back:len(trainPredict)+look\_back, :] = trainPredict # shift test predictions for plotting testPredictPlot = np.empty\_like(dataset) testPredictPlot[:, :] = np.nan testPredictPlot[len(trainPredict)+(look\_back\*2)+1:len(dataset)-1, :] = testPredict # plot baseline and predictions plt.plot(scaler.inverse\_transform(dataset)) plt.plot(trainPredictPlot) plt.plot(testPredictPlot) plt.show()   testFinal = dataset[-look\_back:] testFinal = [pd.DataFrame(testFinal).T.values.tolist()] testFinalPredict = model.predict(testFinal) testFinalPredict = scaler.inverse\_transform(testFinalPredict) print(testFinalPredict)  predict = pd.DataFrame([]) RMSE = pd.DataFrame([]) for index in df:  if index == '销售日期':  continue  # load the dataset  name\_CN = pl.loc[index[:-1]]['单品名称']  print(name\_CN)  dataframe = df[['销售日期',index]]  dataframe.set\_index(["销售日期"], inplace=True)  dataset = dataframe.values  dataset = dataset.astype('float32')  # normalize the dataset  scaler = MinMaxScaler(feature\_range=(0, 1))  dataset = scaler.fit\_transform(dataset)  # split into train and test sets  train\_size = int(len(dataset) \* 0.99)  test\_size = len(dataset) - train\_size  train, test = dataset[0:train\_size,:], dataset[train\_size:len(dataset),:]  # reshape into X=t and Y=t+1  look\_back = 7  trainX, trainY = create\_dataset(train, look\_back)  testX, testY = create\_dataset(test, look\_back)  # reshape input to be [samples, time steps, features]  trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))  testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))  # create and fit the LSTM network  model = Sequential()  model.add(LSTM(30, input\_shape=(1, look\_back)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error', optimizer='adam')  model.fit(trainX, trainY, epochs=30, batch\_size=1, verbose=2)  # make predictions  trainPredict = model.predict(trainX)  testPredict = model.predict(testX)  # invert predictions  trainPredict = scaler.inverse\_transform(trainPredict)  trainY = scaler.inverse\_transform([trainY])  testPredict = scaler.inverse\_transform(testPredict)  testY = scaler.inverse\_transform([testY])  # calculate root mean squared error  trainScore = np.sqrt(mean\_squared\_error(trainY[0], trainPredict[:,0]))  print('Train Score: %.2f RMSE' % (trainScore))  testScore = np.sqrt(mean\_squared\_error(testY[0], testPredict[:,0]))  print('Test Score: %.2f RMSE' % (testScore))  RMSE[name\_CN] = trainScore  # shift train predictions for plotting  trainPredictPlot = np.empty\_like(dataset)  trainPredictPlot[:, :] = np.nan  trainPredictPlot[look\_back:len(trainPredict)+look\_back, :] = trainPredict  # shift test predictions for plotting  testPredictPlot = np.empty\_like(dataset)  testPredictPlot[:, :] = np.nan  testPredictPlot[len(trainPredict)+(look\_back\*2)+1:len(dataset)-1, :] = testPredict  # plot baseline and predictions  plt.plot(scaler.inverse\_transform(dataset))  plt.plot(trainPredictPlot)  plt.plot(testPredictPlot)  plt.show()   testFinal = dataset[-look\_back:]  testFinal = [pd.DataFrame(testFinal).T.values.tolist()]  testFinalPredict = model.predict(testFinal)  testFinalPredict = scaler.inverse\_transform(testFinalPredict)  print(testFinalPredict)  predict[name\_CN] = testFinalPredict[0]   print("--------------------------")  print(RMSE) print(predict)  RMSE.to\_csv('../data/销售量RMSE.csv',sep=',',encoding='utf\_8\_sig') predict.to\_csv('../data/销售量7月1日预估.csv',sep=',',encoding='utf\_8\_sig')  Python 问题三 成本价预处理 import numpy as np import pandas as pd df = pd.read\_excel("../data/附件3.xlsx") df['日期'] = pd.to\_datetime(df['日期']) df = df.set\_index('日期') df['单品编码'] = df['单品编码'].astype(str) print(df)  df\_after = pd.DataFrame([]) df\_after['销售日期']=pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D') df\_after = df\_after.set\_index('销售日期') for index, row in df.iterrows():  df\_after.at[index, row['单品编码']] = row['批发价格(元/千克)'] df\_after = df\_after.fillna(0) print(df\_after) df\_after.to\_csv('../data/成本价按日期单品合并.csv',sep=',',encoding='utf\_8\_sig')  Python 问题三 遗传算法 import numpy as np import matplotlib.pyplot as plt from matplotlib import cm from mpl\_toolkits.mplot3d import Axes3D import pandas as pd   if \_\_name\_\_ == "\_\_main\_\_":  price = pd.read\_csv('../data/单价7月1日预估.csv', encoding='utf-8')  costPrice = pd.read\_csv('../data/成本价7月1日预估.csv', encoding='utf-8')  volume = pd.read\_csv('../data/销售量7月1日预估.csv', encoding='utf-8')  ans = pd.DataFrame([])  ans['P'] = 0  ans['D'] = 0  ans['F'] = 0  for index in price:  if index == '列数':  continue  if index in costPrice and index in volume:  print(index)  print('-'\*30)   DNA\_SIZE = 24  POP\_SIZE = 200  CROSSOVER\_RATE = 0.8  MUTATION\_RATE = 0.005  N\_GENERATIONS = 50  X\_BOUND = [price[index][0] - 1, price[index][0] + 1]  Y\_BOUND = [2.5, 10]  Q = volume[index][0]  C = costPrice[index][0]   def F(x, y):  return x \* 0.5 \* (Q + y - abs(Q - y)) - y \* C    def get\_fitness(pop):  x, y = translateDNA(pop)  pred = F(x, y)  return (pred - np.min(  pred)) + 1e-3  # 减去最小的适应度是为了防止适应度出现负数，通过这一步fitness的范围为[0, np.max(pred)-np.min(pred)],最后在加上一个很小的数防止出现为0的适应度    def translateDNA(pop):  # pop表示种群矩阵，一行表示一个二进制编码表示的DNA，矩阵的行数为种群数目  x\_pop = pop[:, 1::2]  # 奇数列表示X  y\_pop = pop[:, ::2]  # 偶数列表示y   # pop:(POP\_SIZE,DNA\_SIZE)\*(DNA\_SIZE,1) --> (POP\_SIZE,1)  x = x\_pop.dot(2 \*\* np.arange(DNA\_SIZE)[::-1]) / float(2 \*\* DNA\_SIZE - 1) \* (X\_BOUND[1] - X\_BOUND[0]) + \  X\_BOUND[0]  y = y\_pop.dot(2 \*\* np.arange(DNA\_SIZE)[::-1]) / float(2 \*\* DNA\_SIZE - 1) \* (Y\_BOUND[1] - Y\_BOUND[0]) + \  Y\_BOUND[0]  return x, y    def crossover\_and\_mutation(pop, CROSSOVER\_RATE=0.8):  new\_pop = []  for father in pop:  # 遍历种群中的每一个个体，将该个体作为父亲  child = father  # 孩子先得到父亲的全部基因（这里我把一串二进制串的那些0，1称为基因）  if np.random.rand() < CROSSOVER\_RATE:  # 产生子代时不是必然发生交叉，而是以一定的概率发生交叉  mother = pop[np.random.randint(POP\_SIZE)]  # 再种群中选择另一个个体，并将该个体作为母亲  cross\_points = np.random.randint(low=0, high=DNA\_SIZE \* 2)  # 随机产生交叉的点  child[cross\_points:] = mother[cross\_points:]  # 孩子得到位于交叉点后的母亲的基因  mutation(child)  # 每个后代有一定的机率发生变异  new\_pop.append(child)   return new\_pop    def mutation(child, MUTATION\_RATE=0.003):  if np.random.rand() < MUTATION\_RATE: # 以MUTATION\_RATE的概率进行变异  mutate\_point = np.random.randint(0, DNA\_SIZE \* 2) # 随机产生一个实数，代表要变异基因的位置  child[mutate\_point] = child[mutate\_point] ^ 1 # 将变异点的二进制为反转    def select(pop, fitness):  # nature selection wrt pop's fitness  idx = np.random.choice(np.arange(POP\_SIZE), size=POP\_SIZE, replace=True,  p=(fitness) / (fitness.sum()))  return pop[idx]    def print\_info(pop):  fitness = get\_fitness(pop)  max\_fitness\_index = np.argmax(fitness)  print("max\_fitness:", fitness[max\_fitness\_index])  x, y = translateDNA(pop)  print("最优的基因型：", pop[max\_fitness\_index])  print("(x, y):", (x[max\_fitness\_index], y[max\_fitness\_index]))  ans.at[index, 'P'] = x[max\_fitness\_index]  ans.at[index, 'D'] = y[max\_fitness\_index]  ans.at[index, 'F'] = x[max\_fitness\_index] \* 0.5 \* (Q + y[max\_fitness\_index] - abs(Q - y[max\_fitness\_index])) - y[max\_fitness\_index] \* C  print(ans)  fit = []  pop = np.random.randint(2, size=(POP\_SIZE, DNA\_SIZE \* 2))  # matrix (POP\_SIZE, DNA\_SIZE)  for i in range(N\_GENERATIONS):  # 迭代N代  x, y = translateDNA(pop)  pop = np.array(crossover\_and\_mutation(pop, CROSSOVER\_RATE))  # F\_values = F(translateDNA(pop)[0], translateDNA(pop)[1])#x, y --> Z matrix  fitness = get\_fitness(pop)  pop = select(pop, fitness) # 选择生成新的种群  fit.append(np.argmax(fitness))  print(fit)  print\_info(pop)  ans.to\_csv('../data/遗传算法输出.csv', sep=',', encoding='utf\_8\_sig')  Python 问题三 数据预处理 import numpy as np import pandas as pd df = pd.read\_csv("../data/attachments2.csv",encoding='utf-8') df['销售日期'] = pd.to\_datetime(df['销售日期']) df = df.set\_index('销售日期') print(df)  df\_sum = pd.DataFrame([]) df\_sum['销售日期']=pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D') df\_sum = df\_sum.set\_index('销售日期')  df\_count = pd.DataFrame([]) df\_count['销售日期']=pd.date\_range(start='2020-07-01', end='2023-06-30', freq='D') df\_count = df\_count.set\_index('销售日期')  for index, row in df.iterrows():  #print(df.loc[index])  #print(df\_sum.loc[index])  name = str(row['单品编码'])  if name in df\_sum:  df\_sum.at[index, name] += row['销售单价(元/千克)']  df\_count.at[index, name] += 1  else:  df\_sum[name] = 0.0  df\_sum.at[index, name] = row['销售单价(元/千克)']  df\_count[name] = 0.0  df\_count.at[index, name] = 1 print(df\_sum/df\_count) ans = df\_sum/df\_count ans = ans.fillna(0) ans.to\_csv('../data/单价按日期单品合并.csv',sep=',',encoding='utf\_8\_sig')  Python |