## 查看缺失程度

import missingno as msno  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
data = pd.read\_excel(r"..\盈利因子(2023).xlsx")  
  
# 为了显示中文  
from pylab import mpl  
  
mpl.rcParams['font.sans-serif'] = [u'SimHei']  
mpl.rcParams['axes.unicode\_minus'] = False  
  
# 显示空缺值图片  
msno.matrix(data)  
  
plt.show()

## 数据异常值分析

import numpy as np  
import pandas as pd  
def three\_sigma(ser):  
 '''  
 ser参数：被检测的数据，接收DataFrame的一列数据  
 返回：异常值及其对应的行索引  
 '''  
 # 计算平均值  
 mean\_data=ser.mean()  
 # 计算标准差  
 std\_data=ser.std()  
 # 小于μ-3σ或大于μ+3σ的数据均为异常值  
 rule=(mean\_data-3\*std\_data > ser) | (mean\_data+3\*std\_data < ser)  
 # 然后np.arange方法生成一个从0开始，到ser长度-1结束的连续索引，再根据rule列表中的True值，直接保留所有为True的索引，也就是异常值的行索引  
 index=np.arange(ser.shape[0])[rule]  
 # 获取异常值  
 outliers=ser.iloc[index]  
  
 return outliers  
# 读取data.xlsx文件  
excel\_data=pd.read\_excel(r"..\盈利因子(2023).xlsx")  
  
print(excel\_data.head())  
  
excel\_data["销售毛利率"]=excel\_data["销售毛利率"].fillna(excel\_data["销售毛利率"].mean())  
excel\_data["总资产报酬率"]=excel\_data["总资产报酬率"].fillna(excel\_data["总资产报酬率"].mean())  
excel\_data["财务费用率"]=excel\_data["财务费用率"].fillna(excel\_data["财务费用率"].mean())  
excel\_data["息税前利润率"]=excel\_data["息税前利润率"].fillna(excel\_data["息税前利润率"].mean())  
excel\_data["营业总成本率"]=excel\_data["营业总成本率"].fillna(excel\_data["营业总成本率"].mean())  
  
# 对value列进行异常值检测，只要传入一个数据列  
print("销售净利率:",three\_sigma(excel\_data['销售净利率']))  
print("销售毛利率:",three\_sigma(excel\_data['销售毛利率']))  
print("净资产收益率:",three\_sigma(excel\_data['净资产收益率']))  
print("总资产报酬率:",three\_sigma(excel\_data['总资产报酬率']))  
print("财务费用率:",three\_sigma(excel\_data['财务费用率']))  
print("息税前利润率:",three\_sigma(excel\_data['息税前利润率']))  
print("营业总成本率:",three\_sigma(excel\_data['营业总成本率']))

## 箱线图

import pandas as pd  
import matplotlib.pyplot as plt  
# 为了显示中文  
from pylab import mpl  
  
mpl.rcParams['font.sans-serif'] = [u'SimHei']  
mpl.rcParams['axes.unicode\_minus'] = False  
  
df = pd.read\_excel(r"..\盈利因子(2023).xlsx")  
  
df["销售毛利率"] = df["销售毛利率"].fillna(df["销售毛利率"].mean())  
df["总资产报酬率"] = df["总资产报酬率"].fillna(df["总资产报酬率"].mean())  
df["财务费用率"] = df["财务费用率"].fillna(df["财务费用率"].mean())  
df["息税前利润率"] = df["息税前利润率"].fillna(df["息税前利润率"].mean())  
df["营业总成本率"] = df["营业总成本率"].fillna(df["营业总成本率"].mean())  
  
d1 = df['销售净利率']  
d2 = df['销售毛利率']  
d3 = df['净资产收益率']  
d4 = df['总资产报酬率']  
d5 = df['财务费用率']  
d6 = df['息税前利润率']  
d7 = df['营业总成本率']  
  
plt.figure(figsize=(10, 8))  
  
# 绘图  
plt.boxplot(x=[d1,d2,d3,d4,d5,d6,d7],  
 patch\_artist=True,  
 labels=['销售净利率','销售毛利率','净资产收益率','总资产报酬率','财务费用率','息税前利润率','营业总成本率'], # 添加具体的标签名称  
 showmeans=True,  
 boxprops={'color': 'black', 'facecolor': '#9999ff'},  
 flierprops={'marker': 'o', 'markerfacecolor': 'red', 'color': 'black'},  
 meanprops={'marker': 'D', 'markerfacecolor': 'indianred'},  
 medianprops={'linestyle': '--', 'color': 'orange'})  
  
# 显示图形  
plt.show()

## 斯皮尔曼相关系数分析

# 导入 pandas 库，pandas 是一个用于数据操作和分析的库  
import pandas as pd  
  
# 导入 numpy 库，numpy 是一个用于数值计算的库  
import numpy as np  
  
# 导入 seaborn 库，seaborn 是一个基于 matplotlib 的数据可视化库  
import seaborn as sns  
  
# 导入 matplotlib.pyplot 库，这是数据可视化的库  
import matplotlib.pyplot as plt  
  
  
# 定义一个函数，用于读取 Excel 文件中的数据  
# 这个函数接受一个参数 'filename'，代表要读取的 Excel 文件的文件名  
def data(filename):  
 # 使用 pandas 的 read\_excel 函数读取 Excel 文件，将数据保存到 DataFrame 中  
 # df 是 DataFrame 的实例，用于进行后续的数据操作  
 df = pd.read\_excel(filename)  
  
 # 打印读取到的 DataFrame 的内容，方便调试和查看数据是否正确读取  
 print(df)  
  
 # 使用 pandas 的 corr 方法计算 DataFrame 中各列之间的相关性，返回一个相关性矩阵  
 # method='spearman' 表示使用斯皮尔曼等级相关系数计算相关性，这比默认的皮尔逊相关系数更为稳健  
 # 将相关性矩阵保存到 rho 变量中，以备后续使用  
 rho = df.corr(method='spearman')  
 print(rho)  
  
 plt.figure(figsize = (11,11))  
 # 设置 matplotlib 的全局参数，让图形中的字体都使用 SimHei 字体  
 # 这可以让图形中的中文正确显示  
 plt.rcParams['font.family'] = ['SimHei']  
  
 # 设置 matplotlib 的全局参数，让图形中的负号使用正常的字体，而不是默认的斜体  
 plt.rcParams['axes.unicode\_minus'] = False  
  
 # 使用 seaborn 的 heatmap 函数绘制相关性矩阵的热力图  
 # annot=True 表示在热力图上显示数字  
 sns.heatmap(rho, annot=True)  
  
 # 设置图形的标题，使用指定的字体大小  
 plt.title('Heat Map', fontsize=18)  
  
 # 将图形保存为一个 PNG 图片文件，设置图片的 DPI 为 300（每英寸的像素点数为300）  
 #plt.savefig('heatmap1.png', dpi=300)  
  
 # 显示图形，让用户看到绘制的结果  
 plt.show()  
  
  
data(r"..\相对年初增长率.xlsx")

## 主成分分析

# 数据处理  
import pandas as pd  
import numpy as np  
  
# 绘图  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
df = pd.read\_excel(r"..\盈利因子(2023)1.xlsx", index\_col=0).reset\_index(drop=True)  
print(df)  
  
# Bartlett's球状检验  
from factor\_analyzer.factor\_analyzer import calculate\_bartlett\_sphericity  
  
chi\_square\_value, p\_value = calculate\_bartlett\_sphericity(df)  
print(chi\_square\_value, p\_value)  
  
# KMO检验  
# 检查变量间的相关性和偏相关性，取值在0-1之间；KOM统计量越接近1，变量间的相关性越强，偏相关性越弱，因子分析的效果越好。  
# 通常取值从0.6开始进行因子分析  
from factor\_analyzer.factor\_analyzer import calculate\_kmo  
  
kmo\_all, kmo\_model = calculate\_kmo(df)  
print(kmo\_all)  
  
  
# #标准化  
# #所需库  
# from sklearn import preprocessing  
# #进行标准化  
# df = preprocessing.scale(df)  
# print(df)  
  
# #求解系数相关矩阵  
# covX = np.around(np.corrcoef(df.T),decimals=3)  
# print(covX)  
  
# #求解特征值和特征向量  
# featValue, featVec= np.linalg.eig(covX.T) #求解系数相关矩阵的特征值和特征向量  
# print(featValue, featVec)  
  
  
# 不标准化  
# 均值  
def meanX(dataX):  
 return np.mean(dataX, axis=0) # axis=0表示依照列来求均值。假设输入list,则axis=1  
  
  
average = meanX(df)  
print(average)  
  
# 查看列数和行数  
m, n = np.shape(df)  
print(m, n)  
  
# 均值矩阵  
data\_adjust = []  
avgs = np.tile(average, (m, 1))  
print(avgs)  
  
# 去中心化  
data\_adjust = df - avgs  
print(data\_adjust)  
  
# 协方差阵  
covX = np.cov(data\_adjust.T) # 计算协方差矩阵  
print(covX)  
  
# 计算协方差阵的特征值和特征向量  
featValue, featVec = np.linalg.eig(covX) # 求解协方差矩阵的特征值和特征向量  
print(featValue, featVec)  
  
####下面没有区分#######  
  
# 对特征值进行排序并输出 降序  
featValue = sorted(featValue)[::-1]  
print(featValue)  
  
# 绘制散点图和折线图  
# 同样的数据绘制散点图和折线图  
plt.scatter(range(1, df.shape[1] + 1), featValue)  
plt.plot(range(1, df.shape[1] + 1), featValue)  
  
# 显示图的标题和xy轴的名字  
# 最好使用英文，中文可能乱码  
plt.title("Scree Plot")  
plt.xlabel("Factors")  
plt.ylabel("Value")  
  
plt.grid() # 显示网格  
plt.show() # 显示图形  
  
# 求特征值的贡献度  
gx = featValue / np.sum(featValue)  
print(gx)  
  
# 求特征值的累计贡献度  
lg = np.cumsum(gx)  
print(lg)  
  
# 选出主成分  
k = [i for i in range(len(lg)) if lg[i] < 0.97]  
k = list(k)  
print(k)  
  
# 选出主成分对应的特征向量矩阵  
selectVec = np.matrix(featVec.T[k]).T  
selectVe = selectVec \* (-1)  
print(selectVec)  
  
# 主成分得分  
# 标准化第一个参数是df  
# 非标准化第一个参数是data\_adjust  
finalData = np.dot(data\_adjust, selectVec)  
print(finalData)  
  
# 绘制热力图  
  
plt.figure(figsize=(14, 14))  
ax = sns.heatmap(selectVec, annot=True, cmap="BuPu")  
  
# 设置y轴字体大小  
ax.yaxis.set\_tick\_params(labelsize=15)  
plt.title("Factor Analysis", fontsize="xx-large")  
  
# 设置y轴标签  
plt.ylabel("Sepal Width", fontsize="xx-large")  
# 显示图片  
plt.show()

## 普通topsis

import copy  
import pandas as pd  
import numpy as np  
  
result = []  
def function(data1,m):  
 i=0  
 while i<m:  
 X1,X2,X3,X4,X5 = data1[i]  
  
  
 F1 = 0.8951264\*X1 + 0.3993325\*X2 + 0.15154034\*X3 - 0.12748279\*X4 + 0.00812058\*X5  
 F2 = 0.14972227\*X1 - 0.035812\*X2 + 0.04079889\*X3 + 0.98721836\*X4 - 0.00600821\*X5  
 F3 = -0.4148151\*X1 + 0.88460679\*X2 + 0.19063464\*X3 + 0.0873524\*X4 + 0.0377789\*X5  
  
  
 list1 = [F1,F2,F3]  
  
  
 result.append(list1)  
 i=i+1  
  
  
 return result  
  
data=[[1,2,3,4,5],[6,7,8,9,10]]  
  
#计算行数和列数  
data = np.array(data)  
[m,n]=data.shape  
print('行数：',m)  
print('列数：',n)  
  
  
data1 = function(data,m)  
print(data1)  
  
#计算行数和列数  
data1 = np.array(data1)  
[m,n]=data1.shape  
print('行数：',m)  
print('列数：',n)  
  
#数据标准化  
data2=data1.astype('float')  
for j in range(0,n):  
 data2[:,j]=data1[:,j]/np.sqrt(sum(np.square(data1[:,j])))  
#print(data2)  
  
#得到加权后的数据(普通topsis权值为1)  
R=data2\*1  
#得到最大值最小值距离  
r\_max=np.max(R, axis=0) #每个指标的最大值  
r\_min=np.min(R,axis=0) #每个指标的最小值  
d\_z = np.sqrt(np.sum(np.square((R -np.tile(r\_max,(m,1)))),axis=1)) #d+向量  
d\_f = np.sqrt(np.sum(np.square((R -np.tile(r\_min,(m,1)))),axis=1)) #d-向量  
#得到评分  
s=d\_f/(d\_z+d\_f )  
Score=100\*s/max(s)  
for i in range(0,len(Score)):  
 print(f"第{i+1}个投标者百分制得分为：{Score[i]}")

## 熵权法topsis

import numpy as np  
import pandas as pd  
  
#按指定路径导入数据，以“地区”为索引（文件路径需按实际情况更换）  
data = pd.read\_excel(r"..\中特估所有指标打分汇总.xlsx", index\_col = '证券名称')  
print(data)  
  
  
#定义正向指标min-max标准化函数  
def minmax\_p(x):  
 return (x - x.min()) / (x.max() - x.min())  
  
#定义负向指标min-max标准化函数  
def minmax\_n(x):  
 return (x.max() - x) / (x.max() - x.min())  
  
#使用正向指标min-max标准化函数标准化数据  
data\_m = data.apply(minmax\_p, axis = 0)  
print(data\_m)  
  
pij = data\_m / data\_m.sum()  
print(pij)  
  
  
#把pij中的0替换为一个非零的极小值，避免出现ln(0)的警告  
#函数len用于返回对象的长度或元素个数  
pij = pij.replace(0, 1e-100)  
ei = -1 / np.log(len(data\_m)) \* np.sum(pij \* np.log(pij), axis = 0)  
print(ei)  
print('a')  
  
di = 1 - ei  
print(di)  
  
#归一化评价指标的差异系数  
w = di / di.sum()  
print(w)  
  
  
data['熵权法得分'] = data\_m.dot(w)  
print(data)

## 聚类

import pandas as pd  
import numpy as np  
  
data = pd.read\_excel(r"..\全部中特估所有指标打分汇总.xlsx")  
print(data)  
  
import scipy.cluster.hierarchy as sch  
  
A = data.iloc[:, :]  
# A是一个向量矩阵：euclidean代表欧式距离  
distA = sch.distance.pdist(A, metric='euclidean')  
# squareform:将distA数组变成一个矩阵  
distB = pd.DataFrame(sch.distance.squareform(distA.round(2)), columns=data.index, index=data.index)  
print(distB)  
  
import seaborn as sns  
import matplotlib.pyplot as plt  
fig=plt.figure(figsize=(10,10)) #表示绘制图形的画板尺寸为6\*4.5；  
sns.heatmap(distB)  
  
from sklearn.cluster import KMeans  
import matplotlib.pyplot as plt  
  
wgss = []  
for i in range(6):  
 cluster = KMeans(n\_clusters=i + 1, random\_state=0).fit(A)  
 wgss.append(cluster.inertia\_) # inertia\_：每个点到其簇的质心的距离之和。即WGSS  
# 绘制WGSS的碎石图  
print(wgss)  
fig=plt.figure(figsize=(10,10))  
plt.plot([i + 1 for i in range(6)], wgss, marker='o')  
plt.show()  
  
from sklearn.metrics import silhouette\_score #总的聚类效果轮廓系数  
from sklearn.metrics import silhouette\_samples #单个样本的轮廓系数  
silhouette\_scores=[]  
for i in range(1,6):  
 cluster = KMeans(n\_clusters=i+1, random\_state=0).fit(A)  
 # 访问labels\_属性，获得聚类结果  
 y\_pred = cluster.labels\_  
 # 计算平均轮廓系数  
 silhouette\_avg = silhouette\_score(A, y\_pred)  
 silhouette\_scores.append(silhouette\_avg)  
#绘制silhouette\_scores的图形  
print(silhouette\_scores)  
fig=plt.figure(figsize=(10,10))  
plt.plot([i+1 for i in range(1,6)],silhouette\_scores,marker='d')  
plt.show()  
  
  
cluster = KMeans(n\_clusters=3, random\_state=0).fit(A)  
print(cluster.cluster\_centers\_.round(2))  
  
import numpy as np  
#填充区域标签  
bins=[0,1,3,5]  
#填充聚类标签  
data['cluser\_label']=cluster.labels\_ #labels\_返回聚类结果列表  
print(data)  
  
from sklearn.metrics import adjusted\_rand\_score #调整兰德系数  
print('ARI:%s'%(adjusted\_rand\_score(data['cluser\_label'],cluster.labels\_)))  
  
from sklearn.decomposition import PCA  
  
pca = PCA(n\_components=0.95) # 选择方差累积占比95%的主成分  
  
A = data.iloc[:, :]  
pca.fit(A) # 主城分析时每一行是一个输入数据  
result = pca.transform(A) # 计算结果  
fig = plt.figure(figsize=(10, 6)) # 表示绘制图形的画板尺寸为6\*4.5；  
plt.scatter(result[:, 0], result[:, 1], c=data['cluser\_label'], edgecolor='k') # 绘制两个主成分组成坐标的散点图  
plt.show()  
for i in range(result[:, 0].size):  
 plt.text

舆论情感

import pandas as pd  
from snownlp import SnowNLP  
import matplotlib.pyplot as plt  
import seaborn as sns  
from datetime import datetime  
from jieba.analyse import extract\_tags  
import jieba  
from wordcloud import WordCloud  
from collections import Counter  
plt.rcParams['font.sans-serif']=[u'SimHei']  
plt.rcParams['axes.unicode\_minus']=False  
sns.set\_style('whitegrid',{'font.sans-serif':['simhei','Arial']})  
  
baogao = open(r"..\新闻标题名称.txt", encoding = 'UTF-8').read()  
print('文本长度：',len(baogao))  
baogao\_words = [x for x in jieba.cut(baogao) if len(x)>=2]  
c = Counter(baogao\_words).most\_common(20)  
print(c)  
baogao\_words1 = [x for x in jieba.cut\_for\_search(baogao) if len(x)>=2]  
c1 = Counter(baogao\_words1).most\_common(20)  
print(c1)  
  
  
  
  
  
comments = pd.read\_excel(r"..\新闻标题名称.xlsx")  
comments['日期'] = comments['日期'].dt.strftime('%m月%d日')  
print(comments.head())  
  
comments['情绪'] = None  
lencom = len(comments)  
i=0  
while(i<lencom):  
 comments.iloc[i,1]=str(comments.iloc[i,1])[1:7]  
 i+=1  
print(comments.head())  
  
comments['情绪'] = None  
lencom = len(comments)  
i=0  
while(i<lencom):  
 s=SnowNLP(comments.iloc[i,0]).sentiments  
 comments.iloc[i,2]=s  
 i+=1  
print(comments.head())  
  
numbyday = comments['情绪'].groupby(comments['日期']).count()  
emobyday = comments['情绪'].groupby(comments['日期']).sum()  
markbyday = pd.DataFrame()  
markbyday['情绪']=emobyday  
markbyday['计数']=numbyday  
markbyday['情绪指数']=markbyday['情绪']/markbyday['计数']  
print(markbyday.head())  
  
  
plt.figure(figsize=(10,8))  
plt.plot(markbyday['情绪指数'],label='情绪波动')  
plt.title('情绪指数')  
plt.xlabel('交易日期',fontsize=15)  
plt.ylabel('波动率',fontsize=15)  
plt.legend(loc='best')  
plt.show()  
  
lencom = len(comments)  
i=0  
text=''  
while(i<lencom):  
 text = text + comments.iloc[i,0]  
 i+=1  
tags1 = jieba.analyse.extract\_tags(text,topK=100,withWeight=False)  
text1 = ' '.join(tags1)  
wc1 = WordCloud(font\_path='C:\\WINDOWS\\FONTS\\SIMFANG.TTF',  
 background\_color='white',max\_words=100,  
 max\_font\_size=120,min\_font\_size=10,  
 random\_state=42,width=1200,height=900)  
wc1.generate(' '.join([str(item) for item in c1]))  
plt.imshow(wc1)  
plt.axis('off')  
plt.show()

## 时间序列

import sys  
import os  
import warnings  
warnings.filterwarnings("ignore")  
import pandas as pd  
import numpy as np  
from arch.unitroot import ADF  
import matplotlib.pylab as plt  
from matplotlib.pylab import style  
style.use('ggplot')  
import statsmodels.api as sm  
import statsmodels.formula.api as smf  
import statsmodels.tsa.api as smt  
from statsmodels.tsa.stattools import adfuller  
from statsmodels.stats.diagnostic import acorr\_ljungbox  
from statsmodels.graphics.api import qqplot  
pd.set\_option('display.float\_format', lambda x: '%.5f' % x)  
np.set\_printoptions(precision=5, suppress=True)  
"""中文显示问题"""  
plt.rcParams['font.family'] = ['sans-serif']  
plt.rcParams['font.sans-serif'] = ['SimHei']  
  
data = pd.read\_excel(r"..\5支股票收益率.xlsx",parse\_dates=True)  
print(data)  
  
  
#平稳性检验  
print("平稳性检验:\n")  
data["diff1"] = data["中国铝业"].diff(1).dropna()  
data["diff2"] = data["diff1"].diff(1).dropna()  
data1 = data.loc[:,["中国铝业","diff1","diff2"]]  
  
from statsmodels.tsa.stattools import adfuller  
result = adfuller(data['中国铝业'])  
print('ADF Statistic:', result[0])  
print('p-value:', result[1])  
print('Critical Values:', result[4])  
  
data1.plot(subplots=True, figsize=(18, 12),title="差分图")  
plt.show()  
  
print("单位根检验:\n")  
print(ADF(data.diff1.dropna()))  
  
print("白噪声检验:\n")  
from statsmodels.stats.diagnostic import acorr\_ljungbox  
print(acorr\_ljungbox(data.diff1.dropna().values, lags = [i for i in range(4)],boxpierce=True))  
  
#平稳时间序列的自相关图和偏自相关图  
series = data['中国铝业']  
print(series)  
  
# 使用ACF函数 自相关图  
fig, ax = plt.subplots(figsize=(10, 5))  
sm.graphics.tsa.plot\_acf(series, lags=4, ax=ax)  
plt.show()  
  
# 使用PACF函数 偏自相关图  
fig, ax = plt.subplots(figsize=(10, 5))  
sm.graphics.tsa.plot\_pacf(series, lags=1, ax=ax)  
plt.show()  
  
#信息准则定阶：AIC、BIC  
def get\_pq(series):  
 #AIC  
 AIC = sm.tsa.arma\_order\_select\_ic(series, max\_ar=6, max\_ma=4, ic='aic')['aic\_min\_order']  
 #BIC  
 BIC = sm.tsa.arma\_order\_select\_ic(series, max\_ar=6, max\_ma=4, ic='bic')['bic\_min\_order']  
 print('the AIC is{},\nthe BIC is{}\n'.format(AIC,BIC))  
get\_pq(series)  
  
from statsmodels.tsa.arima.model import ARIMA  
# 假设data中有一个名为"exog\_变量的名称"的列，用于外部回归变量  
exog\_variable = data["情绪"]  
# 创建ARIMAX模型，指定AR阶数、差分阶数和MA阶数  
model = ARIMA(data["中国铝业"], order=(4, 1, 1), exog=exog\_variable)  
# 拟合模型并打印摘要信息  
result = model.fit()  
print(result.summary())  
  
# 预测未来值  
forecast\_steps = 3 # 你可以根据需要设置预测步数  
forecast, stderr, conf\_int = result.forecast(steps=forecast\_steps, exog=exog\_variable[-forecast\_steps:])  
print('预测结果：',forecast)  
print('标准误差:',stderr)  
print('标准误差和置信区间:',conf\_int)  
  
# 输出预测曲线  
plt.figure(figsize=(12, 6))  
plt.plot(data['date'],data["中国铝业"], label='实际值', marker='o')  
forecast\_index = pd.date\_range(start='2023-11-03', periods=forecast\_steps+1, closed='right')  
plt.plot(forecast\_index[-3], forecast, label='预测值', marker='x', linestyle='None', color='red') # 将线改为一个点  
plt.xlabel('日期')  
plt.ylabel('中国铝业')  
plt.legend()  
plt.show()  
  
  
resid = result.resid#残差  
fig = plt.figure(figsize=(12,8))  
ax = fig.add\_subplot(111)  
fig = qqplot(resid, line='q', ax=ax, fit=True)  
fig.show()

## 收益率

import matplotlib.pyplot as plt  
import pandas as pd  
from pandas import read\_excel  
import numpy as np  
  
data = pd.read\_excel(r"..\6日收盘价.xlsx")  
print(data)  
  
# 创建空的DataFrame变量，用于存储股票数据  
StockPrices = pd.DataFrame()  
market\_value\_list=[] #存储每支股票的平均市值  
# 创建股票代码的列表  
ticker\_list = ['天地科技', '中国铝业', '中国核电', '中煤能源', '华润三九']  
# 使用循环，挨个获取每只股票的数据，并存储每日收盘价  
for ticker in ticker\_list:  
 stock\_data = read\_excel(r"..\6日收盘价.xlsx", parse\_dates=['日期'], index\_col='日期')  
 stock\_data=stock\_data.loc['2023-10-26':'2023-11-03']  
 StockPrices[ticker] = stock\_data[ticker] #获取每支股票的收盘价  
  
StockPrices.index.name = 'date' # 日期为索引列  
# 输出数据的前5行  
print(StockPrices.head())  
  
  
# 计算每日收益率，并丢弃缺失值  
StockReturns = StockPrices.pct\_change().dropna()  
# 打印前5行数据  
print(StockReturns.head())  
  
StockReturns.to\_excel(r"..\5支股票收益率.xlsx")

## 蒙特卡洛+夏普比率

import matplotlib.pyplot as plt  
import pandas as pd  
from pandas import read\_excel  
import numpy as np  
# 为了显示中文  
from pylab import mpl  
  
mpl.rcParams['font.sans-serif'] = [u'SimHei']  
mpl.rcParams['axes.unicode\_minus'] = False  
  
stock\_return1 = pd.read\_excel(r"..\5支股票收益率1.xlsx")  
print(stock\_return1)  
  
# 计算相关矩阵  
correlation\_matrix = stock\_return1.corr()  
# 输出相关矩阵  
print(correlation\_matrix)  
  
import seaborn as sns  
#创建热图  
sns.heatmap(correlation\_matrix,annot=True,cmap='rainbow',linewidths=1.0,annot\_kws={'size':8})  
plt.xticks(rotation=0)  
plt.yticks(rotation=75)  
plt.show()  
  
# 计算协方差矩阵  
cov\_mat = stock\_return1.cov()  
# 输出协方差矩阵  
print(cov\_mat)  
  
import seaborn as sns  
#创建热图  
sns.heatmap(cov\_mat,annot=True,cmap='rainbow',linewidths=1.0,annot\_kws={'size':8})  
plt.xticks(rotation=0)  
plt.yticks(rotation=75)  
plt.show()  
  
stock\_return = pd.read\_excel(r"..\预测得出5支股票收益率.xlsx")  
print(stock\_return)  
  
# 设置模拟的次数  
number = 30000  
# 设置空的numpy数组，用于存储每次模拟得到的权重、收益率和标准差  
random\_p = np.empty((number, 7))  
# 设置随机数种子，这里是为了结果可重复  
np.random.seed(7)  
  
# 循环模拟30000次随机的投资组合  
for i in range(number):  
 # 生成5个随机数，并归一化，得到一组随机的权重数据  
 random5 = np.random.random(5)  
 random\_weight = random5 / np.sum(random5)  
  
 # 计算年平均收益率  
 mean\_return = stock\_return.mul(random\_weight, axis=1).sum(axis=1).mean()  
 annual\_return = (1 + mean\_return) \*\* 252 - 1  
  
 # 计算年化标准差，也成为波动率  
 random\_volatility = np.sqrt(np.dot(random\_weight.T, np.dot(cov\_mat, random\_weight)))  
  
 # 将上面生成的权重，和计算得到的收益率、标准差存入数组random\_p中  
 random\_p[i][:5] = random\_weight  
 random\_p[i][5] = annual\_return  
 random\_p[i][6] = random\_volatility  
  
# 将Numpy数组转化为DataF数据框  
RandomPortfolios = pd.DataFrame(random\_p)  
# 设置数据框RandomPortfolios每一列的名称  
RandomPortfolios.columns = [ticker + '\_weight' for ticker in stock\_return] + ['Returns', 'Volatility']  
  
# 绘制散点图  
RandomPortfolios.plot('Volatility', 'Returns', kind='scatter', alpha=0.3)  
plt.show()  
  
# 找到标准差最小数据的索引值  
min\_index = RandomPortfolios.Volatility.idxmin()  
  
# 在收益-风险散点图中突出风险最小的点  
RandomPortfolios.plot('Volatility', 'Returns', kind='scatter', alpha=0.3)  
x = RandomPortfolios.loc[min\_index, 'Volatility']  
y = RandomPortfolios.loc[min\_index, 'Returns']  
plt.scatter(x, y, color='red')  
# 将该点坐标显示在图中并保留四位小数  
plt.text(np.round(x, 4), np.round(y, 4), (np.round(x, 4), np.round(y, 4)), ha='left', va='bottom', fontsize=10)  
plt.show()  
  
# 提取最小波动组合对应的权重, 并转换成Numpy数组  
GMV\_weights = np.array(RandomPortfolios.iloc[min\_index, 0:5])  
# 计算GMV投资组合收益  
stock\_return['Portfolio\_GMV'] = stock\_return.mul(GMV\_weights, axis=1).sum(axis=1)  
#输出风险最小投资组合的权重  
print(GMV\_weights)  
  
###  
# 设置无风险回报率为0  
risk\_free = 0  
# 计算每项资产的夏普比率  
RandomPortfolios['Sharpe'] = (RandomPortfolios.Returns - risk\_free) / RandomPortfolios.Volatility  
# 绘制收益-标准差的散点图，并用颜色描绘夏普比率  
plt.scatter(RandomPortfolios.Volatility, RandomPortfolios.Returns, c=RandomPortfolios.Sharpe)  
plt.colorbar(label='Sharpe Ratio')  
plt.show()  
  
# 找到夏普比率最大数据对应的索引值  
max\_index = RandomPortfolios.Sharpe.idxmax()  
# 在收益-风险散点图中突出夏普比率最大的点  
RandomPortfolios.plot('Volatility', 'Returns', kind='scatter', alpha=0.3)  
x = RandomPortfolios.loc[max\_index,'Volatility']  
y = RandomPortfolios.loc[max\_index,'Returns']  
plt.scatter(x, y, color='red')  
#将该点坐标显示在图中并保留四位小数  
plt.text(np.round(x,4),np.round(y,4),(np.round(x,4),np.round(y,4)),ha='left',va='bottom',fontsize=10)  
plt.show()  
  
# 提取最大夏普比率组合对应的权重，并转化为numpy数组  
MSR\_weights = np.array(RandomPortfolios.iloc[max\_index, 0:5])  
# 计算MSR组合的收益  
stock\_return['Portfolio\_MSR'] = stock\_return.mul(MSR\_weights, axis=1).sum(axis=1)  
#输出夏普比率最大的投资组合的权重  
print(MSR\_weights)

print(stock\_return)

## 随机森林求权重

import pandas as pd  
import numpy as np  
from collections import Counter  
import seaborn as sns  
import matplotlib.pyplot as plt  
# 为了显示中文  
from pylab import mpl  
  
mpl.rcParams['font.sans-serif'] = [u'SimHei']  
mpl.rcParams['axes.unicode\_minus'] = False  
  
# 加载数据  
##### 加载训练和测试数据  
#####--------------------------------------------------------------------------------------------------  
#这里读入数据的时候我们没有做任何的处理（像去除空值这些）  
train=pd.read\_excel(r"..\随机森林股票数据.xlsx")  
#查看样本数和特征数  
#train\_num,train\_var\_num=np.shape(train)  
#test\_num,test\_var\_num=np.shape(test)  
#print("训练集：有",train\_num,"个样本","每个样本有",train\_var\_num,"个变量.")  
#print("测试集：有",test\_num,"个样本","每个样本有",test\_var\_num,"个变量.")  
print(train.info())  
#print(test.info())  
  
  
#去除离群点  
#####--------------------------------------------------------------------------------------------------  
#离群点检测  
def detect\_outliers(df,n,features):  
 """  
 输入：  
 df：数据框，为需要检测的样本集  
 n：正整数，样本特征超出四分位极差个数的上限，有这么多个特征超出则样本为离群点  
 features:列表，用于检测是否离群的特征  
 输出：  
  
 """  
 outlier\_indices=[]  
 outlier\_list\_col\_index=pd.DataFrame()  
  
 #对每一个变量进行检测  
 for col in features:  
 #计算四分位数相关信息  
 Q1=np.percentile(df[col],25)  
 Q3=np.percentile(df[col],75)  
 IQR=Q3-Q1  
 #计算离群范围  
 outlier\_step=1.5\*IQR  
 #计算四分位数时如果数据上有空值，这些空值也是参与统计的，所以统计出来的Q1、Q3、IQR这些数据有可能是NAN，但是这并不要紧，在判断是否大于或小于的时候跟NAN比较一定是false，因而样本并不会因为空值而被删除掉  
 #空值会在后面特征工程时再做处理  
  
 #找出特征col中显示的离群样本的索引  
 outlier\_list\_col=df[(df[col]<Q1-outlier\_step)|(df[col]>Q3+outlier\_step)].index  
 #额外存储每一个特征在各样本中的离群判断  
 temp=pd.DataFrame((df[col]<Q1-outlier\_step)|(df[col]>Q3+outlier\_step),columns=[col])  
 #将索引添加到一个综合列表中，如果某个样本有多个特征出现离群点，则该样本的索引会多次出现在outlier\_indices里  
 outlier\_indices.extend(outlier\_list\_col)  
 #额外存储每一个特征在各样本中的离群判断，方便查看数据  
 outlier\_list\_col\_index=pd.concat(objs=[outlier\_list\_col\_index,temp],axis=1)  
 #选出有n个以上特征存在离群现象的样本  
 outlier\_indices=Counter(outlier\_indices)  
 multiple\_outliers=list(k for k,v in outlier\_indices.items() if v>n)  
 return multiple\_outliers,outlier\_list\_col\_index  
  
#获取离群点  
outliers\_to\_drop,outlier\_col\_index=detect\_outliers(train,2,["行业评分","组织形式评分","盈利因子评分","研发因子得分","风险因子得分","成长因子得分","年化平均收益率"])  
#输出离群点信息  
  
print(train.loc[outliers\_to\_drop])  
print(outlier\_col\_index.loc[outliers\_to\_drop])#查看哪个特征对样本成为离群点有决定作用.  
  
#输出数据集各变量详细信息  
print(train.describe())  
  
#删除离群点  
train = train.drop(outliers\_to\_drop, axis = 0).reset\_index(drop=True)  
#整合训练集和测试集（只是为了后面在有的内容统计和值的处理上更方便，也可以不整合对每个数据集单独处理）  
#整合需要在训练集剔除离群点后再做，因为测试集是不需要剔除离群点  
  
  
#查看缺失值  
print(train.info())  
  
#输出数据集各变量详细信息  
print(train.describe())  
  
#查看数据之间的相关性  
#corr中无参数默认是皮尔逊相关系数，若要改成斯皮尔曼相关系数要在corr中加上method='spearman'  
plt.figure(figsize=(9,10))  
g=sns.heatmap(train[["行业评分","组织形式评分","盈利因子评分","研发因子得分","风险因子得分","成长因子得分","年化平均收益率"]].corr(method='spearman'),annot=True,fmt = ".2f",cmap = "coolwarm")  
plt.show()  
  
#查看是否有缺失值  
print(train.info())  
  
#重新获取训练数据和测试数据  
train=train[:]  
train["年化平均收益率"]=train["年化平均收益率"].astype(int)  
Y\_train=train["年化平均收益率"]  
X\_train=train.drop(labels=["年化平均收益率"],axis=1)  
test=train[:]  
test.drop(labels=["年化平均收益率"],axis=1,inplace=True)  
  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import GridSearchCV  
from sklearn.model\_selection import KFold  
# 搜索随机森林的最佳参数  
RFC = RandomForestClassifier()  
kf = KFold(n\_splits=5)  
## 设置参数网络  
rf\_param\_grid = {"max\_depth": [None],  
 "max\_features": [1, 3, 10],  
 "min\_samples\_split": [2, 3, 10],  
 "min\_samples\_leaf": [1, 3, 10],  
 "bootstrap": [False],  
 "n\_estimators" :[100,300],  
 "criterion": ["gini"]}  
gsRFC = GridSearchCV(RFC,param\_grid = rf\_param\_grid, cv=kf, scoring="accuracy", n\_jobs= 1, verbose = 1)  
gsRFC.fit(X\_train,Y\_train)  
RFC\_best = gsRFC.best\_estimator\_  
print(RFC\_best)  
# 打印最佳得分  
print(gsRFC.best\_score\_)  
  
from sklearn.model\_selection import learning\_curve  
# 效果评估  
#####--------------------------------------------------------------------------------------------------  
### 效果评估之学习曲线  
def plot\_learning\_curve(estimator, title, X, y, ylim=None, cv=None,  
 n\_jobs=1, train\_sizes=np.linspace(.1, 1.0, 5)):  
 """Generate a simple plot of the test and training learning curve"""  
 plt.figure(figsize=(15,15))  
 plt.title(title)  
 if ylim is not None:  
 plt.ylim(\*ylim)  
 plt.xlabel("Training examples")  
 plt.ylabel("Score")  
 train\_sizes, train\_scores, test\_scores = learning\_curve(  
 estimator, X, y, cv=cv, n\_jobs=n\_jobs, train\_sizes=train\_sizes)  
 train\_scores\_mean = np.mean(train\_scores, axis=1)  
 train\_scores\_std = np.std(train\_scores, axis=1)  
 test\_scores\_mean = np.mean(test\_scores, axis=1)  
 test\_scores\_std = np.std(test\_scores, axis=1)  
 plt.grid()  
  
 plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,  
 train\_scores\_mean + train\_scores\_std, alpha=0.1,  
 color="r")  
 plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,  
 test\_scores\_mean + test\_scores\_std, alpha=0.1, color="g")  
 plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r",  
 label="Training score")  
 plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g",  
 label="Cross-validation score")  
  
 plt.legend(loc="best")  
 return plt  
g = plot\_learning\_curve(gsRFC.best\_estimator\_,"RF mearning curves",X\_train,Y\_train,cv=kf)  
plt.show()  
  
# 特征变量权重分析  
#####--------------------------------------------------------------------------------------------------  
nrows = ncols = 1  
fig, axes = plt.subplots(nrows = nrows, ncols = ncols, sharex="all", figsize=(15,15))  
#names\_classifiers = [("AdaBoosting", ada\_best),("ExtraTrees",ExtC\_best),("RandomForest",RFC\_best),("GradientBoosting",GBC\_best)]  
names\_classifiers = [("RandomForest",RFC\_best)]  
nclassifier = 0  
for row in range(nrows):  
 for col in range(ncols):  
 name = names\_classifiers[nclassifier][0]  
 classifier = names\_classifiers[nclassifier][1]  
 indices = np.argsort(classifier.feature\_importances\_)[::-1][:40]  
 g = sns.barplot(y=X\_train.columns[indices][:40],x = classifier.feature\_importances\_[indices][:40] , orient='h')  
 g.set\_xlabel("Relative importance",fontsize=12)  
 g.set\_ylabel("Features",fontsize=12)  
 g.tick\_params(labelsize=9)  
 g.set\_title(name + " feature importance")  
 nclassifier += 1  
plt.show()

print(X\_train.columns[indices])  
print(classifier.feature\_importances\_[indices])

## 随机森林预测

# 导入必要的库  
import pandas as pd  
import numpy as np  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score  
  
# 读取数据集  
train = pd.read\_excel(r"..\随机森林股票数据.xlsx") # 假设数据集保存在名为train.csv的CSV文件中  
train = train.iloc[:, 1:]  
test = pd.read\_excel(r"..\5支股票各项得分用于随机森林数据.xlsx")  
test = test.iloc[:,1:]  
print('train:',train)  
print('test:',test)  
  
  
# 划分训练集和测试集  
X\_train = train.drop(labels=['年化平均收益率'],axis=1)  
y\_train = train['年化平均收益率']  
test = test[:]  
  
# 构建随机森林回归模型  
rf = RandomForestRegressor(n\_estimators=100, random\_state=42,bootstrap=False, max\_features=10, min\_samples\_split=3) # 设置决策树的数量为100  
  
# 训练模型  
rf.fit(X\_train, y\_train)  
  
# 预测结果  
y\_pred = rf.predict(test)  
  
test["预测值"] = y\_pred  
  
# 模型评估  
#mse = mean\_squared\_error(y\_train, y\_pred)  
#mae = mean\_absolute\_error(y\_train, y\_pred)  
#r2 = r2\_score(y\_train, y\_pred)  
  
#print('Mean Squared Error (MSE):', mse)  
#print('Mean Absolute Error (MAE):', mae)  
#print('R-squared (R2):', r2)  
print('预测值的数量：', len(y\_pred))  
print('预测值：',y\_pred)  
print(test)  
test.to\_excel(r"..\5支股票随机森林预测值.xlsx")