附录

附录1：Autocorrelation\_coefficient-1.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

double a[5520][105]; //预处理后的数据

double ans[5520][105]; //answer

int main(){

memset(ans,0,sizeof(ans));

freopen("Autocorrelation\_coefficient\_in.txt","r",stdin);

freopen("Autocorrelation\_coefficient\_out.txt","w",stdout);

int tar[16]={4,7,11,17,20,21,23,41,47,57,65,71,72,73,76,96};

//编号从1开始

for(int i=1;i<=5519;i++){

for(int j=1;j<=100;j++)scanf("%lf",&a[i][j]);

}

//读入数据

for(int J=1;J<=100;J++){

//处理第J列数据

int c=1;

double sum=0,y=0;

//sum 用于计算前c个和，以便求平均数

for(;c<=15;c++){

sum+=a[c][J];

}

//从十六开始

for(;c<=5519;c++){

sum+=a[c][J];

y=sum/c;

//求出分母上的var\*c

double deno=0;

for(int i=1;i<=c;i++){

deno+=( a[i][J] - y )\*( a[i][J] - y );

}

for(int k=1;k<=15;k++){

double tmp=0;

//tmp保留当前k延迟条件下的自相关系数

for(int i=1;i<=c-k;i++){

tmp+=(a[i][J]-y)\*(a[i+k][J]-y)/deno;

}

ans[c][J]+=tmp/15;

}

}

}

for(int i=1;i<=5519;i++){

for(int j=1;j<=100;j++){

printf("%.10lf\t",ans[i][j]);

}

printf("\n");

}

return 0;

}

附录2：Autocorrelation\_coefficient-2.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

double a[5760][105]; //预处理后的数据

double ans[5760][105]; //answer

int main(){

memset(ans,0,sizeof(ans));

freopen("Autocorrelation\_coefficient\_3\_in.txt","r",stdin);

freopen("Autocorrelation\_coefficient\_3\_out\_100.txt","w",stdout);

int tar[16]={4,7,11,17,20,21,23,41,47,57,65,71,72,73,76,96};

//编号从1开始

for(int i=1;i<=5759;i++){

for(int j=1;j<=100;j++)scanf("%lf",&a[i][j]);

}

//读入数据

for(int J=1;J<=100;J++){

//处理第J列数据

int c=1;

double sum=0,y=0;

for(;c<=100;c++){

sum+=a[c][J];

}

//sum 用于计算前c个和，以便求平均数

//从十六开始

for(c=101;c<=5759;c++){

sum=sum+a[c][J]-a[c-100][J];

y=sum/100;

//求出分母上的var\*c

double deno=0;

for(int i=c-99;i<=c;i++){

deno+=( a[i][J] - y )\*( a[i][J] - y );

}

for(int k=1;k<=15;k++){

double tmp=0;

//tmp保留当前k延迟条件下的自相关系数

for(int i=c-99;i<=c-k;i++){

tmp+=(a[i][J]-y)\*(a[i+k][J]-y)/deno;

}

ans[c][J]+=tmp/15;

}

}

}

for(int i=1;i<=5759;i++){

for(int j=1;j<=100;j++){

printf("%.10lf\t",abs( ans[i][j] ));

}

printf("\n");

}

return 0;

}

附录3：SW\_test数据类型转换.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

//int find(const char \*s, int pos = 0) const;//从pos开始查找字符串s在当前串中的位置

//string substr(int pos = 0,int n = npos) const;//返回pos开始的n个字符组成的字符串

int main(){

string s;

freopen("SW\_test\_in.txt","r",stdin);

freopen("SW\_test\_out.txt","w",stdout);

for(int i=0;i<100;i++){

getline(cin,s);

string s1,s2,s3;

int a1=s.find('=');

int a2=s.find(',');

s1=s.substr(a1,a2-a1);

cout<<s1;

printf("\t");

s=s.substr(a2);

a1=s.find('=');

a2=s.find(')');

s1=s.substr(a1,a2-a1);

cout<<s1<<endl;

}

return 0;

}

附录4：test1.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF=0x3f3f3f3f;

const long long mod=100000007;

const double e=2.718281828459045;

const double pi=3.1415926535;

#define CK cout<<"OK\n";

int main(){

int a95=00,a90=0,a85=0,a80=0;

freopen("1.txt","r",stdin);

freopen("output1.txt","w",stdout);

for(int i=0;i<100;i++){

double a;

cin>>a;

if(a>0.95)a95++;

if(a>0.90)a90++;

if(a>0.85)a85++;

if(a>0.80)a80++,cout<<i+1<<endl;

}

cout<<a95<<endl<<a90<<endl<<a85<<endl<<a80<<endl;

return 0;

}

附录5：第2题.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

double ans[5520][105]; //answer

struct A{

double s1,s2,s3;

double score;

int rank;

friend bool operator<(A x,A y){

return x.score>y.score;

}

A(double s1=0,double s2=0,double s3=0,double score=0,int r=0):s1(s1),s2(s2),s3(s3),score(score),rank(r){}

}a[5520];

int main(){

freopen("score\_2\_in.txt","r",stdin);

freopen("score\_2\_out.txt","w",stdout);

for(int i=1;i<=5519;i++){

for(int j=1;j<=100;j++)scanf("%lf",&ans[i][j]);

}

double p1=1.151329,p2=3.0625,p3=1.331716;

for(int i=1;i<=5520;i++){

a[i].s1=a[i].s2=a[i].s3=0;

//分别计算s1 s2 s3 的平均值

a[i].s1=( ans[i][4]+ans[i][17] )/2;

a[i].s2=(

+ans[i][7]

+ans[i][20]

+ans[i][21]

+ans[i][41]

+ans[i][65]

+ans[i][72]

+ans[i][73]

+ans[i][76]

+ans[i][71]

+ans[i][96]

+ans[i][23]

+ans[i][47]

+ans[i][57]

)/13;

a[i].s3=( ans[i][11]+ans[i][94] )/2;

a[i].score=p1\*a[i].s1 + p2\*a[i].s2 + p3\*a[i].s3;

a[i].rank=i;

}

sort(a+1,a+5520);

for(int i=1;i<=5;i++){

printf("%d\t%.8lf\n",a[i].rank,a[i].score);

}

return 0;

}

附录6：第3题.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

double ans[5760][105]; //answer

struct A{

double s1,s2,s3;

double score;

int rank;

friend bool operator<(A x,A y){

return x.score>y.score;

}

A(double s1=0,double s2=0,double s3=0,double score=0,int r=0):s1(s1),s2(s2),s3(s3),score(score),rank(r){}

}a[5760];

int main(){

freopen("Autocorrelation\_coefficient\_3\_out\_100.txt","r",stdin);

freopen("score\_3\_out.txt","w",stdout);

for(int i=1;i<=5759;i++){

for(int j=1;j<=100;j++)scanf("%lf",&ans[i][j]);

}

double p1=1.151329,p2=3.0625,p3=1.331716;

for(int i=1;i<=5759;i++){

a[i].s1=a[i].s2=a[i].s3=0;

//分别计算s1 s2 s3 的平均值

a[i].s1=( ans[i][4]+ans[i][17] )/2;

a[i].s2=(

+ans[i][7]

+ans[i][20]

+ans[i][21]

+ans[i][41]

+ans[i][65]

+ans[i][72]

+ans[i][73]

+ans[i][76]

+ans[i][71]

+ans[i][96]

+ans[i][23]

+ans[i][47]

+ans[i][57]

)/13;

a[i].s3=( ans[i][11]+ans[i][94] )/2;

a[i].score=p1\*a[i].s1 + p2\*a[i].s2 + p3\*a[i].s3;

a[i].rank=i;

}

sort(a+5520,a+5580);

printf("%d\t%.8lf\n",a[5520].rank,a[5520].score);

sort(a+5580,a+5640);

printf("%d\t%.8lf\n",a[5580].rank,a[5580].score);

sort(a+5640,a+5700);

printf("%d\t%.8lf\n",a[5640].rank,a[5640].score);

sort(a+5700,a+5760);

printf("%d\t%.8lf\n",a[5700].rank,a[5700].score);

return 0;

}

附录7：ARIMA计算.m

clear

%% 数据读取

DATA = xlsread('matlab1.xlsx')

fin = zeros(300,100)

pq=zeros(2,100)

load H\_safe.mat H

%%

for i = 99:100

y=DATA(:,i);

T = length(y);

%H里保存了差分的次数（10的差0次）

if (H(i)==1)

d=1

else

d=0

end

tar=10000

tar\_p=1

tar\_q=1

for p=1:5

for q=1:5

mdl = arima(p, q, d);

EstMdl = estimate(mdl, y);

res = infer(EstMdl,y);

tmp=var(res)

aic=log(tmp)+170\*(p+q+1)/5519

if (tar>aic)

tar=aic

tar\_p=p

tar\_q=q

end

end

end

mdl = arima(tar\_p, tar\_q, d);

EstMdl = estimate(mdl, y);

res = infer(EstMdl,y);

[yF,yMSE] = forecast(EstMdl,300,'Y0',y);

UB = yF + 1.96\*sqrt(yMSE); % 95%置信区间上限

LB = yF - 1.96\*sqrt(yMSE); % 95%置信区间下限

fin(:,i) = yF

pq(1,i)=tar\_p

pq(2,i)=tar\_q

end

附录8：ARIMA枚举求参.m

clear

%% ARIMA(p,d,q)

DATA = xlsread('matlab1.xlsx')

fin = zeros(300,100)

%%

y = DATA(:,1);

T = length(y);

tar=10000

tar\_p=0

tar\_q=0

for p=1:5

for q=1:5

mdl = arima(p, q, 0);

EstMdl = estimate(mdl, y);

res = infer(EstMdl,y);

tmp=var(res)

aic=log(tmp)+170\*(p+q+1)/5519

if (tar>aic)

tar=aic

tar\_p=p

tar\_q=q

end

end

end

mdl = arima(tar\_p, tar\_q, 0);

EstMdl = estimate(mdl, y);

res = infer(EstMdl,y);

[yF,yMSE] = forecast(EstMdl,300,'Y0',y);

UB = yF + 1.96\*sqrt(yMSE); % 95%置信区间上限

LB = yF - 1.96\*sqrt(yMSE); % 95%置信区间下限

figure(6)

h4 = plot(y,'Color',[.75,.75,.75],'LineWidth',2);

hold on

h5 = plot(T+1:T+300,yF,'r','LineWidth',2);

h6 = plot(T+1:T+300,UB,'k--','LineWidth',1.5);

plot(T+1:T+300,LB,'k--','LineWidth',1.5);

h7 = gca;

plot((T+0.5)\*[1;1], h7.YLim, 'k--','LineWidth',1.5), % 样本点与预测点的时间分割线

h7.XLim = [0,T+302];

legend([h4,h5,h6],'CPI','预测值',...

'95置信区间','Location','Northwest')

title('CPI预测值')

hold off

fin(:,1) = yF

%%

y = DATA(:,2);

T = length(y);

tar=10000

tar\_p=0

tar\_q=0

for p=1:5

for q=1:5

mdl = arima(p, q, 0);

EstMdl = estimate(mdl, y);

res = infer(EstMdl,y);

tmp=var(res)

aic=log(tmp)+170\*(p+q+1)/5519

if (tar>aic)

tar=aic

tar\_p=p

tar\_q=q

end

end

end

mdl = arima(tar\_p, tar\_q, 0);

EstMdl = estimate(mdl, y);

res = infer(EstMdl,y);

[yF,yMSE] = forecast(EstMdl,300,'Y0',y);

UB = yF + 1.96\*sqrt(yMSE); % 95%置信区间上限

LB = yF - 1.96\*sqrt(yMSE); % 95%置信区间下限

figure(6)

h4 = plot(y,'Color',[.75,.75,.75],'LineWidth',2);

hold on

h5 = plot(T+1:T+300,yF,'r','LineWidth',2);

h6 = plot(T+1:T+300,UB,'k--','LineWidth',1.5);

plot(T+1:T+300,LB,'k--','LineWidth',1.5);

h7 = gca;

plot((T+0.5)\*[1;1], h7.YLim, 'k--','LineWidth',1.5), % 样本点与预测点的时间分割线

h7.XLim = [0,T+302];

legend([h4,h5,h6],'CPI','预测值',...

'95置信区间','Location','Northwest')

title('CPI预测值')

hold off

fin(:,2) = yF

附录9：ARIMA作图.m

clear

%% ARIMA(p,d,q)消费指数时间序列数据

% Australian Consumer Price Index (CPI) measured from 1972 and 1991,

load Data\_JAustralian

y = DataTable.PAU;

T = length(y);

%% 平稳性的单位根检验

% adftest

[h, p] = adftest(y)

% pptest

[h, p] = pptest(y)

%% 差分

diff\_y = diff(y); % 一阶差分运算

%% 平稳性的单位根检验

[h, p] = adftest(diff\_y) % 平稳性adf检验

%% autocorr

figure(3), autocorr(diff\_y); %自相关系数图

%% parcorr

figure(4), parcorr(diff\_y); %偏自相关系数图

%% 建立选定的较优模型ARIMA(2,1,0)

mdl = arima(2, 1, 0);

EstMdl = estimate(mdl, y);

%% 残差检验

% 可视化分析

res = infer(EstMdl,y);

%% Ljung-Box Q (lbq) 检验

[h, p] = lbqtest(res)

%% 预测

[yF,yMSE] = forecast(EstMdl,20,'Y0',y);

UB = yF + 1.96\*sqrt(yMSE); % 95%置信区间上限

LB = yF - 1.96\*sqrt(yMSE); % 95%置信区间下限

figure(6)

h4 = plot(y,'Color',[.75,.75,.75],'LineWidth',2);

hold on

h5 = plot(T+1:T+20,yF,'r','LineWidth',2);

h6 = plot(T+1:T+20,UB,'k--','LineWidth',1.5);

plot(T+1:T+20,LB,'k--','LineWidth',1.5);

h7 = gca;

plot((T+0.5)\*[1;1], h7.YLim, 'k--','LineWidth',1.5), % 样本点与预测点的时间分割线

h7.XLim = [0,T+22];

legend([h4,h5,h6],'CPI','预测值',...

'95置信区间','Location','Northwest')

title('CPI预测值')

hold off

附录10：H\_safe.mat

（由于该数据不能直接用txt格式展示，故此处只展示其数值）

0 1 1 1 0 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 1 0 0 1 0 0 0 1 1 1 1 1 0 0 1 0 0 1 1 10 0 0 0 0 1 0 1 1 0 10 1 1 0 1 1 0 0 0 10 1 1 0 1 1 0 0 0 1 1 1 1 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 1 0 10 1 1

附录11：第4题计算.py

#!/usr/bin/env python  
# coding: utf-8  
  
# In[175]:  
  
  
import numpy as np  
import pandas as pd  
  
  
# In[176]:  
  
  
data = pd.read\_excel('全时段数据.xlsx')  
  
  
# In[177]:  
  
  
data.drop(data.columns[0], axis=1, inplace=True)  
data  
  
  
# In[178]:  
  
  
data\_np=np.array(data)  
  
  
# In[179]:  
  
  
AC = pd.read\_excel('Autocorrelation\_coefficient\_100.xlsx')  
  
  
# In[180]:  
  
  
AC.drop(AC.columns[0], axis=1, inplace=True)  
  
  
# In[181]:  
  
  
AC  
  
  
# In[182]:  
  
  
ac\_np=np.array(AC)  
ac\_np.shape  
  
  
# In[183]:  
  
  
type2=np.array([1,2,3,10,15,16,18,19,30,31,32,34,38,39,54,56,60,63,69,74,79,81,84,85,86,87,89,91,92,97,99])  
type3=np.array([4,7,11,17,20,21,23,41,47,57,65,71,72,73,76,96])  
  
  
# In[184]:  
  
  
type3.shape[0]  
  
  
# In[185]:  
  
  
type2.shape[0]  
  
  
# ### 数据导入完成  
  
# In[186]:  
  
  
score1=np.zeros(48\*100).reshape(48,100)  
score1.shape  
  
  
# In[187]:  
  
  
data\_ave=np.mean(data\_np,axis=0)  
data\_var=np.var(data\_np,axis=0)  
data\_ave.shape[0]  
  
  
# In[188]:  
  
  
n=5760  
  
  
# In[189]:  
  
  
data\_var=data\_var\*n  
  
  
# In[190]:  
  
  
data\_div=np.zeros(n\*100).reshape(n,100)  
  
  
# In[191]:  
  
  
import math   
  
  
# data\_power 表示正态间的权重关系  
  
# In[192]:  
  
  
data\_power=np.zeros(100)  
for i in range(100):  
    data\_power[i]=math.sqrt(data\_var[i])/data\_ave[i]  
  
  
# In[193]:  
  
  
for I in range(type2.shape[0]):  
    J=type2[I]  
    for  i in range(n):  
        data\_div[i,J]=(data\_np[i,J]-data\_ave[J])\*(data\_np[i,J]-data\_ave[J])/(data\_var[J]+0.00001)  
data\_div  
  
  
# data\_div 表示一个时刻正态的分数  
  
# In[194]:  
  
  
data\_score=np.zeros(48\*100).reshape(48,100)  
data\_score.shape  
  
  
# In[195]:  
  
  
for j in range(100):  
    for i in range(48):  
        for k in range(60):  
            c=i\*60+k  
            data\_score[i,j]=data\_score[i,j]+data\_div[c,j]  
  
  
# In[196]:  
  
  
data\_score  
  
  
# ### 正态分布处理完成  
  
# In[197]:  
  
  
ac\_score=np.zeros(48\*100).reshape(48,100)  
data\_score.shape  
  
  
# In[198]:  
  
  
for j in range(100):  
    for i in range(48):  
        sum=0  
        for k in range(60):  
            c=i\*60+k  
            sum=sum+ac\_np[c,j]  
        sum=sum/60  
        ac\_score[i,j]=sum  
  
  
# In[208]:  
  
  
tmp2=pd.DataFrame(ac\_score)  
tmp2.to\_excel('ac\_score.xlsx')  
  
  
# In[200]:  
  
  
data\_score=data\_score\*48  
  
  
# ### 自相关处理完成  
  
# In[202]:  
  
  
data\_power\_sum=np.sum(data\_power)  
  
  
# In[203]:  
  
  
score\_fin=np.zeros(48)  
  
  
# In[204]:  
  
  
for i in range(48):  
    for J in range(type2.shape[0]):  
        j=type2[J]  
        score1[i,j]=data\_score[i,j]  
        score\_fin[i]=score\_fin[i]+data\_score[i,j]\*data\_power[j]/data\_power\_sum\*31  
        print(i,j,data\_score[i,j],data\_power[j])  
    for J in range(type3.shape[0]):  
        j=type3[J]  
        score1[i,j]=ac\_score[i,j]  
        score\_fin[i]=score\_fin[i]+ac\_score[i,j]  
  
  
# In[205]:  
  
  
#tmp2=pd.DataFrame(data\_power)  
#tmp2.to\_excel('data\_power.xlsx')  
  
  
# In[206]:  
  
  
score\_fin  
  
  
# In[211]:  
  
  
score2=100-score\_fin  
  
  
# In[212]:  
  
  
S=(score2-np.min(score2))/( np.max(score2)-np.min(score2) )\*100  
tmp2=pd.DataFrame(S)  
tmp2.to\_excel('S.xlsx')  
  
  
# In[ ]:

附录12：灵敏度测试.py

#!/usr/bin/env python  
# coding: utf-8  
  
# In[1]:  
  
  
import numpy as np  
import pandas as pd  
  
  
# In[2]:  
  
  
data = pd.read\_excel('全时段数据.xlsx')  
  
  
# In[3]:  
  
  
data.drop(data.columns[0], axis=1, inplace=True)  
data  
  
  
# In[4]:  
  
  
data\_np=np.array(data)  
  
  
# In[5]:  
  
  
AC = pd.read\_excel('Autocorrelation\_coefficient\_100.xlsx')  
  
  
# In[6]:  
  
  
AC.drop(AC.columns[0], axis=1, inplace=True)  
  
  
# In[7]:  
  
  
AC  
  
  
# In[8]:  
  
  
ac\_np=np.array(AC)  
ac\_np.shape  
  
  
# In[9]:  
  
  
type2=np.array([1,2,3,10,15,16,18,19,30,31,32,34,38,39,54,56,60,63,69,74,79,81,84,85,86,87,89,91,92,97,99])  
type3=np.array([4,7,11,17,20,21,23,41,47,57,65,71,72,73,76,96])  
  
  
# In[10]:  
  
  
type3.shape[0]  
  
  
# In[11]:  
  
  
type2.shape[0]  
  
  
# ### 数据导入完成  
  
# In[12]:  
  
  
score1=np.zeros(48\*100).reshape(48,100)  
score1.shape  
  
  
# In[13]:  
  
  
data\_ave=np.mean(data\_np,axis=0)  
data\_var=np.var(data\_np,axis=0)  
data\_ave.shape[0]  
  
  
# In[14]:  
  
  
n=5760  
  
  
# In[15]:  
  
  
data\_var=data\_var\*n  
  
  
# In[16]:  
  
  
data\_div=np.zeros(n\*100).reshape(n,100)  
  
  
# In[17]:  
  
  
import math   
  
  
# data\_power 表示正态间的权重关系  
  
# In[18]:  
  
  
data\_power=np.zeros(100)  
for i in range(100):  
    data\_power[i]=math.sqrt(data\_var[i])/data\_ave[i]  
  
  
# In[19]:  
  
  
for I in range(type2.shape[0]):  
    J=type2[I]  
    for  i in range(n):  
        data\_div[i,J]=(data\_np[i,J]-data\_ave[J])\*(data\_np[i,J]-data\_ave[J])/(data\_var[J]+0.00001)  
data\_div  
  
  
# data\_div 表示一个时刻正态的分数  
  
# In[20]:  
  
  
data\_score=np.zeros(48\*100).reshape(48,100)  
data\_score.shape  
  
  
# In[21]:  
  
  
for j in range(100):  
    for i in range(48):  
        for k in range(60):  
            c=i\*60+k  
            data\_score[i,j]=data\_score[i,j]+data\_div[c,j]  
  
  
# In[22]:  
  
  
data\_score  
  
  
# ### 正态分布处理完成  
  
# In[23]:  
  
  
ac\_score=np.zeros(48\*100).reshape(48,100)  
data\_score.shape  
  
  
# In[24]:  
  
  
for j in range(100):  
    for i in range(48):  
        sum=0  
        for k in range(60):  
            c=i\*60+k  
            sum=sum+ac\_np[c,j]  
        sum=sum/60  
        ac\_score[i,j]=sum  
  
  
# In[25]:  
  
  
tmp2=pd.DataFrame(ac\_score)  
tmp2.to\_excel('ac\_score.xlsx')  
  
  
# In[26]:  
  
  
data\_score=data\_score\*48  
  
  
# ### 自相关处理完成  
  
# In[27]:  
  
  
data\_power\_sum=np.sum(data\_power)  
  
  
# In[28]:  
  
  
score\_fin=np.zeros(48)  
  
  
# In[29]:  
  
  
for i in range(48):  
    for J in range(type2.shape[0]):  
        j=type2[J]  
        score1[i,j]=data\_score[i,j]  
        score\_fin[i]=score\_fin[i]+data\_score[i,j]\*data\_power[j]/data\_power\_sum\*31  
        print(i,j,data\_score[i,j],data\_power[j])  
    for J in range(type3.shape[0]):  
        j=type3[J]  
        score1[i,j]=ac\_score[i,j]  
        score\_fin[i]=score\_fin[i]+ac\_score[i,j]\*0.33333  
  
  
# In[30]:  
  
  
#tmp2=pd.DataFrame(data\_power)  
#tmp2.to\_excel('data\_power.xlsx')  
  
  
# In[31]:  
  
  
score\_fin  
  
  
# In[32]:  
  
  
score2=100-score\_fin  
  
  
# In[33]:  
  
  
S=(score2-np.min(score2))/( np.max(score2)-np.min(score2) )\*100  
tmp2=pd.DataFrame(S)  
tmp2.to\_excel('S.xlsx')  
  
  
# In[ ]:

附录13：数据审视.py

#!/usr/bin/env python  
# coding: utf-8  
  
# In[3]:  
  
  
import numpy as np  
import pandas as pd  
data = pd.read\_excel('附件1.xlsx', index\_col=1, header=0, parse\_dates=True)  
data  
  
  
# In[4]:  
  
  
data.drop(data.columns[0], axis=1, inplace=True)  
  
  
# In[5]:  
  
  
data\_np=np.array(data)  
  
  
# 已经转为np类型了。  
#   
  
# In[6]:  
  
  
# np.mean(a, axis=0) # axis=0，计算每一列的均值  
  
  
# In[7]:  
  
  
# 求出平均值  
data\_ave=np.mean(data\_np,axis=0)  
  
  
# In[8]:  
  
  
# 求出方差  
data\_var=np.var(data\_np,axis=0)  
data\_var  
  
  
# 第一步预处理，先用处理离群点的方法处理冲击性数据  
  
# In[9]:  
  
  
import copy  
  
  
# In[10]:  
  
  
for i in range(100):  
    slct=data\_np[:,i]  
    a1=np.percentile(slct,25,interpolation='midpoint')  
    a2=np.percentile(slct,50,interpolation='midpoint')  
    a3=np.percentile(slct,75,interpolation='midpoint')  
    up\_lim=a3\*2-a2  
    down\_lim=a1\*2-a2  
    for j in range(5519):  
        if (data\_np[j][i]>up\_lim or data\_np[j][i]<down\_lim):  
            data\_np[j][i]=a2  
  
  
# 第二步预处理，处理符合正态分布的列  
  
# In[11]:  
  
  
import scipy.stats as stats  
  
  
# In[12]:  
  
  
for i in range(100):  
    if (data\_var[i]<1e-4):  
        for j in range(5519):  
            data\_np[j][i]=data\_ave[i]  
  
  
# In[13]:  
  
  
ans1=np.zeros(300).reshape(3,100)  
for i in range(100):  
    slct=data\_np[:,i]  
    sta,pvalue=stats.shapiro(slct)  
    ans1[0,i]=sta  
    ans1[1,i]=pvalue  
    if (sta>0.9):  
        ans1[2:i]=1  
  
  
# In[14]:  
  
  
ans2=pd.DataFrame(ans1)  
ans2.to\_excel('A.xlsx')  
  
  
# In[ ]:  
  
  
  
  
  
# In[ ]:

附录14：数据预处理.py

#!/usr/bin/env python  
# coding: utf-8  
  
# In[20]:  
  
  
import numpy as np  
import pandas as pd  
data = pd.read\_excel('data.xlsx', index\_col=1, header=0, parse\_dates=True)  
data  
  
  
# In[21]:  
  
  
data.drop(data.columns[0], axis=1, inplace=True)  
  
  
# In[22]:  
  
  
data\_np=np.array(data)  
  
  
# 已经转为np类型了。  
#   
  
# In[23]:  
  
  
# np.mean(a, axis=0) # axis=0，计算每一列的均值  
  
  
# In[24]:  
  
  
# 求出平均值  
data\_ave=np.mean(data\_np,axis=0)  
  
  
# In[25]:  
  
  
# 求出方差  
data\_var=np.var(data\_np,axis=0)  
data\_var  
  
  
# 第一步预处理，先用处理离群点的方法处理冲击性数据  
  
# In[26]:  
  
  
import copy  
  
  
# In[27]:  
  
  
for i in range(100):  
    slct=data\_np[:,i]  
    a1=np.percentile(slct,25,interpolation='midpoint')  
    a2=np.percentile(slct,50,interpolation='midpoint')  
    a3=np.percentile(slct,75,interpolation='midpoint')  
    up\_lim=a3\*2-a2  
    down\_lim=a1\*2-a2  
    for j in range(5519):  
        t=0  
        if (data\_np[j][i]>up\_lim or data\_np[j][i]<down\_lim):  
            data\_np[j][i]=a2  
            t=1  
    if(t==1):  
        print(i)  
              
  
  
# 第二步预处理，处理符合正态分布的列  
  
# In[19]:  
  
  
ans2=pd.DataFrame(data\_np)  
ans2.to\_excel('A.xlsx')  
  
  
# In[ ]:

附录15：原始数据作图.py

#!/usr/bin/env python

# coding: utf-8

# In[1]:

# -\*- coding: utf-8 -\*-

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.graphics.tsaplots import plot\_acf

from matplotlib.pyplot import acorr

from scipy.stats import t

def drawing(data):

'''

画出数据随着时间的变化图

'''

cols = data.columns

time\_series = range(1,data.shape[0]+1)

for col in cols:

fig = plt.figure()

plt.plot(time\_series, data[col])

plt.xlabel('Time Index', fontsize=16)

plt.ylabel('Value', fontsize=16)

plt.title(f'Sensor\_{col}', fontsize=20)

fig.savefig(f'传感器{col}.png')

plt.grid()

plt.show()

# In[2]:

def save\_medium(verbose=True):

'''

数据的统计描述，并保存

'''

print('数据的统计信息如下： \n', data.describe())

data\_summary = data.describe()

data\_summary.to\_excel('数据统计描述.xlsx')

# In[3]:

data = pd.read\_excel('附件1(Appendix 1)2021-51MCM-Problem C.xlsx', index\_col=1, header=0, parse\_dates=True)

# 删除第一列（时间编号）

data.drop(data.columns[0], axis=1, inplace=True)

# In[4]:

a = data.iloc[:, 5].autocorr(lag=1)

save\_medium(verbose=True)

# 画出传感器读数-时间图

drawing(data)

# In[ ]: