针对异常检测与预警问题的数据挖掘模型

摘要

本文针对风险性异常评估及预测问题，基于某生产企业某 23h 生产区域的仪器设备记录的时间序列数据集，采用多种验证方式相结合，进行风险性异常波动判断，利用线性模型，将验证过程的相关分析数据加权求和，得到量化的风险性异常程度分值。将得到的异常程度分值利用线性回归模型、rbf 核方法回归模型和多项式回归模型进行时序上的拟合预测，选择效果最佳的多项式回归模型，得到风险性异常预警模型。最后，套用风险性异常程度评分模型和风险性异常预警模型的异常程度得分，反向映射到 $[0, 100]$ 范围内形成安全评价评分模型。

针对问题一，首先对 100 个传感器进行筛选，通过统计每个传感器测量得到的数值个数以及可视化的折线图，将无波动的传感器进行剔除。再进行冲击性风险检验，利用算法 A 找出冲击性数据并进行冲击性风险评估。其次利用正态分布和均匀分布检验进行独立性和偶发性检验，判断波动是否为正常性波动。最后利用自相关系数进行联动性和持续性的检验，判断是否为风险性异常波动。

针对问题二，在问题一的判别方法基础上，利用冲击性风险数据、高斯分布分析数据和自相关系数，根据线性模型进行加权求和，得到量化的风险性异常程度分值。其中由于自相关系数与风险性异常波动的判定直接相关且更加可靠，其权重也更重。

针对问题三，在问题二的风险性异常程度数据基础上，将数据的采样间隔放大到 15min，保证数据的时间序列不可分割性以及拟合预测时的效率。再进行不同模型的拟合测试，剔除过拟合与欠拟合的模型，将效果相对最好的多项式回归模型作为风险性异常预警模型。

针对问题四，在问题二和问题三的基础上，利用问题三预估的当日最后两个小时一场得分数据结果，套用问题二中的风险性异常程度评分模型，得到全天所有时段时间间隔为 30min 的异常程度得分，并将其反向映射得到安全评分，形成安全评价评分模型。

关键词：异常程度判定；冲击性判定；KS 检验；自相关性；时序回归分析与 5 预测。

1 问题重述

1.1 问题背景

推动生产企业高质量发展，最根本的底线是保证安全、防范风险，而生产过程中产生的数据能够实时反映潜在的风险。利用现代科学技术对生产中产生的海量数据进行分析，找出其中蕴含的规律以及隐藏的重要信息，能够对生产活动以及相关策略安排提供有力科学的依据与帮助。

1.2 题目所给信息及参数

题目所给附件一为某生产企业某日 00:00:00-22:59:59 由生产区域的仪器设备记录的时间序列数据（已经进行数据脱敏），包含 100 个传感器在对应时间区间内每隔 15s 的数据记录，每一时刻均有对应编号。在本题中并未给出数据的具体名称。这些数据可能是温度、浓度、压力等与安全密切相关的数据。

1.3 所需解决的问题

根据附件一以及相关知识，需要解决以下四个问题：

1. 附件 1 所给出的数据都可能存在波动，且所有波动都在安全值范围内。请建立数学模型，给出判定非风险性异常数据和风险性异常数据的方法。

2. 结合问题 1 的结果，建立数学模型，给出风险性异常数据异常程度的量化评价方法，要求使用百分制（0-100 分）对每个时刻数据异常程度进行评价（分值越高表示异常程度越高）。应用所建立的模型和附件 1 的数据，找到数据中异常分值最高的 5 个时刻及这 5 个时刻对应的异常传感器编号（每个时刻只填写 5 个异常程度最高的传感器编号，异常传感器不足 5 个则无需填满；如果得分为 0，可以不用填写异常传感器编号），并给出数学模型对所得结果进行评价。

3. 为了提前发现未来生产过程中可能存在的风险隐患，请建立风险性异常预警模型，预测当日 23:00:00-23:59:59 可能产生的风险性异常。结合问题 2 中给出的风险性异常程度量化评价方法，指出 23:00:00-23:59:59 中四个时间段（见表 2），每个时间段内的最高异常分值及对应的异常传感器编号（只填写 5 个异常程度最高的传感器编号，异常传感器不足 5 个则无需填满；如果得分为 0，可以不用填写异常传感器编号）；

4. 根据问题 2 和问题 3 中的结果，建立数学模型对该生产企业整个生产系统的安全性进行评价，请在 00:00:00-23:59:59 内每隔 30 分钟，用 0-100 分进行安全性评分，0 分表示安全性最低，100 分表示安全性最高（包括 00:00:00-23:00:00 的得分和 23:00:00-23:59:59 的预测得分），并用适当的方法对所给评分的结果进行评价和敏感性分析。。

2 问题分析

2.1 问题一分析

对于本题，非风险性异常数据与风险性异常数据的判定是整题的基石。合理有效的判定方法可以更加科学地区分这两种异常数据，并为接下来的量化分析提供良好的依据。

非风险性异常指具有规律性、独立性、偶发性等特点的正常性波动，并不产生安全风险，如随着外界温度或者产量变化的波动，或者可能是传感器误报。这些波动不需要人为干预。风险性异常指

具有持续性、联动性等特点的异常性波动，这些波动的出现时生产过程中的不稳定因素造成的，预示着潜在的安全隐患，需要人为的干预、分析以及评定风险等级。

本题需要对每一传感器在 00:00:00-22:59:59 时段内的数据进行异常分析，找出其中的异常波动，并结合传感器之间的异常波动，判断哪些是非风险性异常，哪些是风险性异常 [1]。经过综合分析，我们的希望建立的数据挖掘模型从如下几方面来对数据进行分析与信息提取：

1. 筛选传感器：考虑到传感器数量高达 100 个，并且不同传感器在该时间段内表现不同，我们需要首先筛选出可能产生异常性波动的传感器来简化问题，提高效率，避免未产生异常性波动的传感器对分析判别产生影响。

2. 冲击性风险判定：对于某些正常性波动中可能出现一些冲击性数据，我们需要判断这些数据属于白噪音中正常的波动还是来自生产过程的冲击性风险。因此我们采用了来自 GB/T 6379.5 的算法 A 来进行冲击性风险判定，统计冲击性风险的数据含量。

3. 独立性和偶发性判断：正常性波动具有独立性和偶发性。正态分布检验与均匀分布检验可以验证这些波动是否属于正常的白噪音，量化误差判断是否属于风险，从而推出这些波动是否属于正常性波动。

4. 持续性和联动性风险：风险性异常数据具有持续性和联动性的特点。自相关系数是判断这两者的重要依据。我们通过自相关系数的相关计算来判断该波动数据是否属于风险性异常数据。

2.2 问题二分析

问题二需要在问题一的风险性异常波动判定基础上，对风险性的异常波动程度进行定量的评估。思路如下：

1. 获得了问题一中的步骤 2 冲击性检验的冲击性程度、步骤 3 独立性和偶发性判断的高斯分布分析数据和步骤 4 持续性和联动性中的验证数据；

2. 将上述三种数据利用线性模型进行加权求和，得到量化的风险性异常数值。

2.3 问题三分析

在问题三中，我们需要用到问题二中获得前 23h 风险性异常程度数据。思路如下：

1. 首先将问题二中获得的数据进行采样间隔为 15min 的重新采样，以同时保证时间序列的不可分割性以及拟合预测时的效率；

2. 尝试线性模型与非线性模型，比较不同模型的拟合预测效果，选择相对最佳的作为风险性异常预警模型。

2.4 问题四分析

问题四需要建立数学模型，对该生产企业整个生产系统的安全性进行评价，即以 30 分钟为单位，对生产系统给出一个最高分为 100、最低分为 0 的安全分数。对此，我们采取了以下步骤：

1. 模型首先以 30 分钟为一个单位间隔，根据问题二中建立的量化评价风险性异常数据异常程度的数学模型，对原始数据和问题三中得到的预测数据，计算出 46 行、100 列风险性得分；

2. 再将同一时段的 100 列传感器得分求和，得到每一时段总得分；

3. 最后利用 Max-Min 标准化即离散标准化的方法，将得分值映射到 $[0,100]$ 之间，得到生产系统每一段时段的总安全分数。

3 模型假设

1. 在问题二估算异常程度得分时，模型认为所有传感器贡献的异常程度权重相同，即默认传感器无重要性之分。

4 符号说明

表 1: 工厂生产风险性异常波动判定与预测模型符号表

符号	说明
m	无权重的冲击性风险值
$m^{'}$	有权重的冲击性风险值
$risk_1$	有权重的冲击性风险
A^2	AD 检验的检验统计量
X_i	次序统计量
AD	A^2 的调整
D_n	KS 检验的检验统计量
$F_n(x)$	经验分布函数
F(x)	实际分布函数
$I_{[-\infty,X]}(X_i)$	指示函数
W	SW 检验的检验统计量
r	自相关系数

5 模型建立与问题求解

5.1 数据处理与变量分析

5.2 问题一分析

5.2.1 思路分析

本题的重点在于分析判断传感器中的异常数据的特征，并对其进行判断是否为风险性波动。我们从冲击性风险、独立性和偶发性风险以及持续性和联动性风险三个方面进行判别。对于正常的测量数据而言，往往会存在一些冲击性数据，这些数据产生了波动，但在一定范围内并不属于异常，也不总是代表着潜在的风险，通过冲击性风险检验可以将这些数据剔除。对于具有独立性和偶发性的波动，可以将这些数据视为非风险性异常数据，我们通过正态分布和均匀分布检验来进行判别。最后，利用自相关系数进行相关性检验，判别波动数据的持续性和联动性，进一步确定其是否属于风险性异常数据。

思路如图所示：

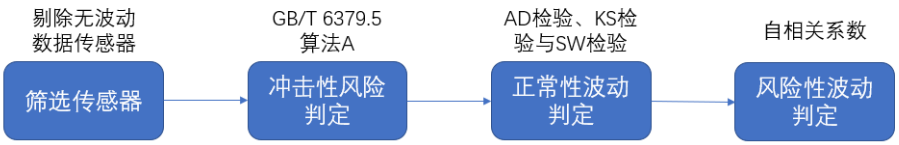

图 1: **问题一思路流程**

5.2.2 数据预处理

我们需要对 100 个传感器进行初步筛选，将没有产生波动、一直稳定正常输出的传感器筛选出去，减轻后续分析工作量。

我们统计每一个传感器在 00:00:00-22:59:59 时段内一共输出了多少个不同的数值，并对每一个传感器进行了可视化处理，如图所示：

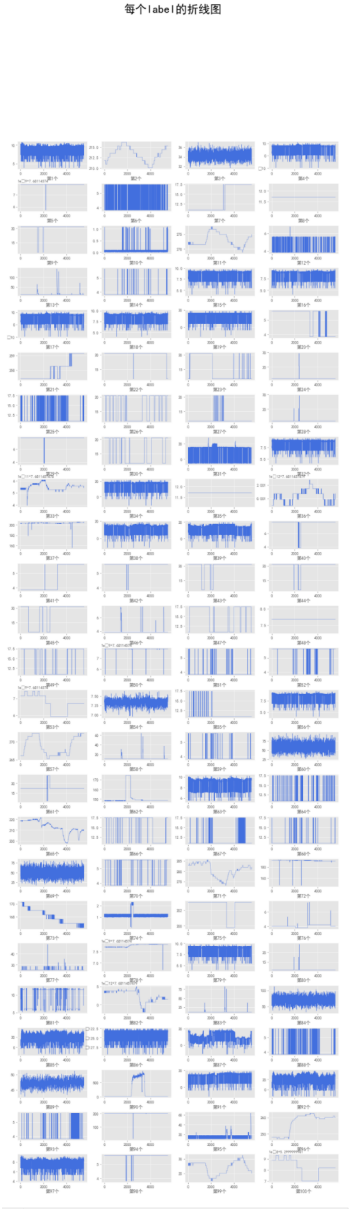

图 2: 100 **个传感器数据折线图**

从图中我们可以具体看出，有些传感器的折线图是完全平滑的直线，有些是具有冲击性的结果，而有些是白噪声的典型表现。剩下的数据图则是风险性异常的传感器。我们首先筛选掉完全平滑的数据所对应的传感器。即第 8，第 35 和第 48 个传感器。因为其可能对白噪声和自相关性的分析产生影响。通过可视化传感器数据的折线图，我们可以很直观的对整体的数据有一个大概的了解，并且有助于我们验证下面算法以及筛选过程的正确性。

5.2.3 模型建立

首先，对于出现了波动的传感器，我们进行了冲击性数据分析。对于一组数据而言，出现合理范围内的冲击性波动属于正常的白噪音，我们需要判别在筛选步骤中出现的波动是否属于冲击性数据。这里我们采用了来自 GB/T 6379.5 的算法 A，计算数据平均值和标准差的稳健值，统计冲击性风险的数据含量。

算法 A 的计算过程如下：

首先计算初始值：

$$x^* = med\, x_i \tag{1}$$

$$s^* = 1.483 \times med\,|x_i - x^*| \tag{2}$$

对每一个 x_i，有：

$$x_i^* = \begin{cases} x^* - \delta & x_i < x^* - \delta \\ x^* + \delta & x_i > x^* + \delta \\ x_i & else \end{cases} \tag{3}$$

其中：$\delta = 1.5s^*$，再次计算：

$$x^* = \sum \frac{x_i^*}{p} \tag{4}$$

$$s^* = 1.134\sqrt{\sum \frac{(x_i^* - x^*)^2}{(p-1)}} \tag{5}$$

重复：

$$x_i^* = \begin{cases} x^* - \delta & x_i < x^* - \delta \\ x^* + \delta & x_i > x^* + \delta \\ x_i & else \end{cases} \tag{6}$$

直到 s^* 的第三位有效数字和 x^* 的对应数字在连续两次迭代中不变。

因此，对感应器的数据，我们通过算法 A 后，得到最终的 x_i^*，记取值为 $x^* - \delta$ 或 $x^* + \delta$ 的数量为 m，冲击性风险为 $\frac{m}{n}$。

然而我们发现这种情况所得出来的筛选结果并不是很理想，因此我们考虑到由于冲击性持续时间越长，意味着风险越大，因此使用添加权重的方法。我们将连续的个数与其所在连续个数中的位置赋上 1,2，……，m 的权重，并将每个序列最终权重之和，记为该序列的冲击性风险值 m'。

所以，冲击性风险为：

$$risk_1 = \frac{m'}{n} \tag{7}$$

如图为采用了无权重与有权重两种方式的冲击性检验：

In [146]:

```
x_star_A=[]
s_star_A=[]
A_risk=[[]for i in range(100)]

#x中取值为上下界的个数几位m，冲击性风险=m/n
for i in range(2,102):
    x=data.iloc[:,i]
    x, xstar, sstar, higherbound, lowerbound = perform_algorithm_A(x)
    x_star_A.append(xstar)
    s_star_A.append(sstar)
    #print(i,": x_star=",xstar,"s_star=",sstar)
    n=len(x)
    m=0
    for j in range(n):
        if x[j]==higherbound or x[j]==lowerbound:
            m+=1
    risk_temp=m/n
    A_risk[i-2].append(i-1)
    A_risk[i-2].append(risk_temp)
    print(A_risk[i-2])
```

```
[1, 0.08950896901612611]
[2, 0.06758470737452436]
[3, 0.13408226127921724]
[4, 0.22721507519478165]
[5, 0.19931146946910672]
[6, 0.09512592861025548]
[7, 0.4312375430331582]
[8, 1.0]
[9, 0.023736184091320892]
[10, 0.020655915926798333]
[11, 0.11723138249682914]
```

图 3: **无权重的冲击性检验**

In [148]:

```python
#带权重的A_algorithm
A_r=[]
for i in range(2,102):
    x=data.iloc[:,i]
    x, xstar, sstar, higherbound, lowerbound = perform_algorithm_A(x)
    x=x.tolist()
    m=cal_risk(x,higherbound,lowerbound)
    n=len(x)
    A_r.append(m/n)
    print(i-1,":",m/n)
#print(A_r)
```

```
1 : 0.14857764087697048
2 : 12.638340279036058
3 : 0.19115781844537053
4 : 0.5144047834752673
5 : 91.14459141148758
6 : 0.12846530168508788
7 : 485.71589055988403
```

图 4: **有权重的冲击性检验**

最终得出，在以下传感器中出现了冲击性数据：

```
[5, 7, 8, 20, 21, 23, 24, 26, 27, 28, 29, 30, 35, 37, 40, 41, 42, 47, 48,
49, 52, 53, 57, 62, 66, 70, 90, 94, 96, 98, 100]
```

图 5: **出现冲击性数据的传感器编号**

接下来，我们进行了独立性和偶发性误差的检验。对于满足正态分布或均匀分布的时间序列，我们可以认为是正常的。我们分别进行了 AD 检验（Anderson–Darling test）、KS 检验（Kolmogorov–Smirnov test）和 SW 检验（Shapiro–Wilk test）。通过比较三种检验的结果，我们最终采用了 KS 检验的结果。

AD 检验的检验统计量为：

$$A^2 = -N - \frac{1}{n}\sum_{i-1}^{n}(2i-1)(ln\Phi(Y_i) + ln(1-\Phi(Y_{N-1-i}))) \tag{8}$$

其中 Y_i 为：

$$Y_i = \frac{X_i - \hat{\mu}}{\hat{\sigma}} \tag{9}$$

均值和方差有：$\hat{\mu} = \mu$，$\hat{\sigma}^2 = \sigma^2$

X_i 为次序统计量，即将原始数据根据升序的方式排序。当检验统计量 A^2 大于临界值，则说明不服从正态分布。对于本题中样本数量很大，可近似看成 ∞ 时，可以利用近似公式计算 p-value，若 p-value 越接近 1，则意味着样本越有可能是正态分布，即风险越小。

$$p-value = \begin{cases} exp(1.2937 - 5.709AD - 0.0186AD^2) & AD \geq 0.6 \\ exp(0.9177 - 4.279AD - 1.38AD^2) & AD \geq 0.34 \\ exp(-8.318 + 42.796AD - 59.937AD^2 & AD \geq 0.2 \\ exp(-13.436 + 101.14AD - 223.73AD^2) & otherwise \end{cases} \tag{10}$$

其中 AD 为 A^2 的调整：

$$AD = A^2 \times (1 + \frac{0.75}{n} + \frac{2.25}{n^2}) \tag{11}$$

我们还尝试了 KS 检验 [2]。KS 检验的检验统计量为：

$$D_n = \sup_x |F_n(x) - F(x)| \tag{12}$$

其中 $F_n(x)$ 是经验分布，F(x) 为实际分布，如下:

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{[-\infty,x]}(X_i) \tag{13}$$

$I_{[-\infty,x]}(X_i)$ 是指示函数，若有样本落在范围内，则为 1. 经验函数根据样本分布，模拟了总体的分布函数。$\sup_x$ 为上确界。根据 Glivenko–Cantelli 原理，若样本 x_i 来源于总体分布 F(x)，则 D_n 将收敛于 0。与 AD 检验同理，计算 p-value，若 p-value 越大，则数据服从均匀分布，风险越小。

除此之外，我们还进行了 SW 检验 [4]。SW 检验用于验证一个随机样本数据是否来自正态分布。

检验统计量 W 为:

$$W = \frac{(\sum a_i y_i)^2}{\sum (y_i - \bar{y})^2} \tag{14}$$

其中 $\mathrm{a} = (\mathrm{a}_1, ..., \mathrm{a}_\mathrm{n})^\mathrm{T}$ 符合条件: $(\sum a_i y_i)^2$ 是 $(n-1)\sigma^2$ 的最佳线性无偏估计 (best linear unbiased estimate,BLUE[]),σ 是样本来自的正态分布的标准差。

其中 a 的确切值为:

$$\mathrm{a} = (\mathrm{m}^\mathrm{T}\mathrm{V}^{-1}\mathrm{V}^{-1}\mathrm{m})^{-\frac{1}{2}}\mathrm{m}^\mathrm{T}\mathrm{V}^{-1} \tag{15}$$

其中矩阵 V 是协方差矩阵，属于 n 个标准正态分布的随机变量的顺序统计量，m 是这些变量的期望组成的向量。

W 的分母为 $(n-1)\sigma^2$ 的一个无偏估计。对于来自正态分布的数据而言，W 的分子分母均会趋向一个常数: $(n-1)\sigma^2$ 的估计值。

获得统计量后，设置显著性水平 α 为 0.05,。与 AD 检验和 KS 检验同理，利用 p-value 进行判断。若 p-value 大于 α，则验证样本来自正态分布。

但由于 SW 检验对于样本数量超过 5000 的数据集检验效果并不理想，因此我们在尝试之后舍弃了该种方法。

```
E:\anaconda\envs\pytorchsrtp\lib\site-packages\scipy\stats\morestats.py:1681: UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

图 6: SW **检验不适用于本题**

使用三种方法得到的 p_value 值如下图所示。由数据可以发现，正态分布检验、均匀分布检验等对风险较为敏感，p_value 大多为零，如 KS 检验共有 74 列数据为 0（不符合检验要求），AD 检验则有 77 列为 0。

	kstest	shapiro	normaltest
0	2.498080e-253	0.000000e+00	2.509996e-129
1	1.685362e-65	1.416266e-39	1.889383e-120
2	2.704795e-03	5.011055e-04	5.041140e-02
3	0.000000e+00	0.000000e+00	0.000000e+00
4	0.000000e+00	0.000000e+00	0.000000e+00
...	...	...	...
95	0.000000e+00	0.000000e+00	0.000000e+00
96	2.971576e-252	0.000000e+00	3.799018e-151
97	0.000000e+00	0.000000e+00	0.000000e+00
98	1.310008e-233	0.000000e+00	0.000000e+00
99	3.502775e-249	0.000000e+00	0.000000e+00

100 rows × 3 columns

图 7: **三种验证方式得出的** p-value

```
bdf[bdf['normaltest']==0]
```

	kstest	shapiro	normaltest
3	0.000000e+00	0.0	0.0
4	0.000000e+00	0.0	0.0
5	0.000000e+00	0.0	0.0
6	0.000000e+00	0.0	0.0
8	0.000000e+00	0.0	0.0
...	...	...	...
94	0.000000e+00	0.0	0.0
95	0.000000e+00	0.0	0.0
97	0.000000e+00	0.0	0.0
98	1.310008e-233	0.0	0.0
99	3.502775e-249	0.0	0.0

77 rows × 3 columns

(a)

```
bdf[bdf['kstest']==0]
```

	kstest	shapiro	normaltest
3	0.0	0.0	0.0
4	0.0	0.0	0.0
5	0.0	0.0	0.0
6	0.0	0.0	0.0
8	0.0	0.0	0.0
...	...	...	...
92	0.0	0.0	0.0
93	0.0	0.0	0.0
94	0.0	0.0	0.0
95	0.0	0.0	0.0
97	0.0	0.0	0.0

74 rows × 3 columns

(b)

图 8: KS **与** AD **中** p-value **为** 0 **的项**

综合三种验证方式，为了使不同传感器数据之间具有更多的风险区分度，我们选择 KS 检验方法作为检验传感器数据独立性和偶发性的方法。

最后，我们利用自相关系数来判断传感器数据是否具备联动性和持续性，以此来判断波动数据是否属于风险性异常数据。

自相关系数旨在计算当前时刻，和前 k 个时刻的 Pearson 相关系数，其计算公式如下 [5]：

$$r = \frac{\sum_{i=k+1}^{n}(x_t - \bar{x}_t)(x_{t-k} - \bar{x}_{t-k}}{\sqrt{\sum_{i=k+1}^{n}(x_t - \bar{x}_t)(x_(t-k) - \bar{x}_(t-k)^2}} \tag{16}$$

若 r 越接近于 1 或-1，则意味着数据之间的线性相关性越强，即风险性异常概率越大。
如图为部分传感器在不同 k 取值时的自相关关系：

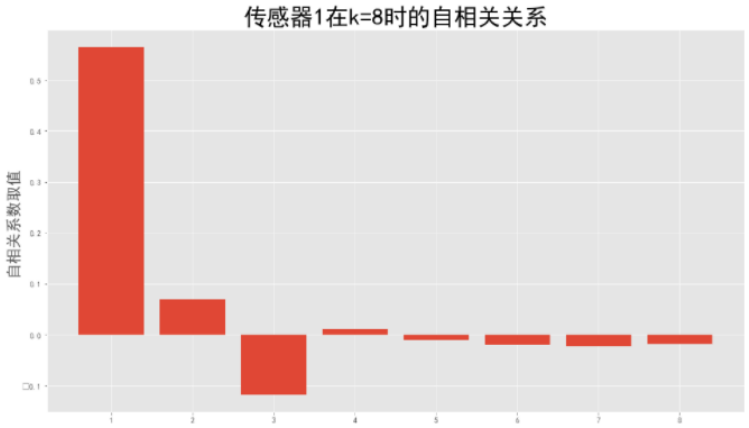

图 9: **传感器 1 在 k=8 时的自相关关系**

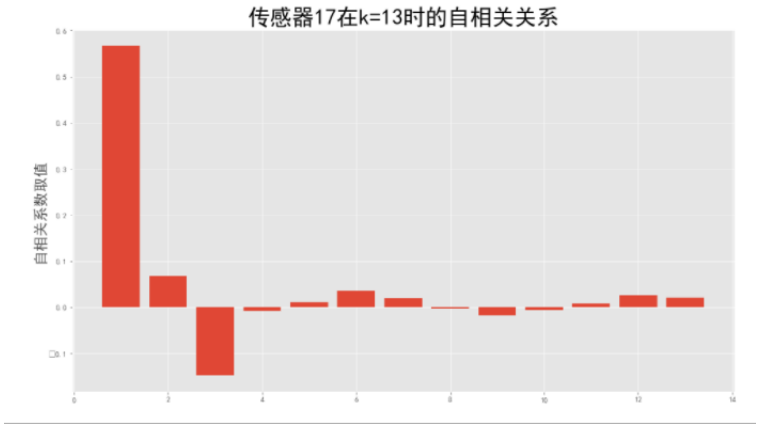

图 10: **传感器 17 在 k=13 时的自相关关系**

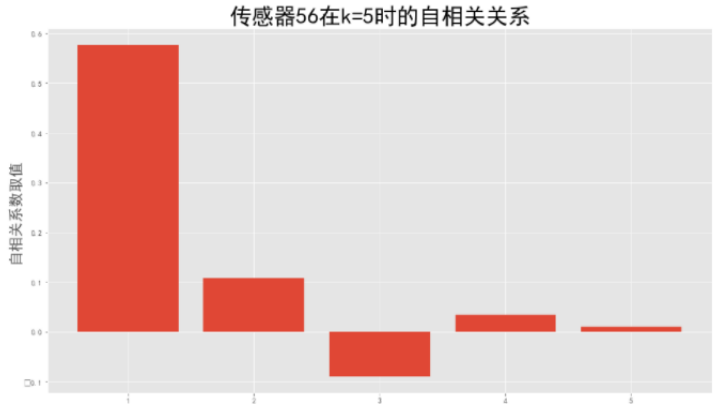

图 11: **传感器 56 在 k=5 时的自相关关系**

5.2.4 模型评价与推广

我们尽可能充分考虑了每个传感器的各种可能性，将对于风险判定无关的传感器剔除。同时，利用多种检验方式，如冲击性验证、AD 检验、KS 检验和 SW 检验，对其进行非风险性判定。最后利用自相关度进行持续性和联动性判定其是否为风险性异常数据。我们尽可能充分考虑风险性异常数据和非风险性异常数据的各种特点，并结合每种验证方式的适用场景，进行了科学有效的验证判别。

5.3 问题二分析

5.3.1 思路分析

第二问要求结合问题一的结果，建立数学模型，给出风险性异常数据异常程度的量化评价标准。考虑到时间序列过于紧凑以及序列不可分、数据量大的性质，我们从对 0 点 23 点，以 30 分钟为单位进行了传感器总风险的计算。通过问题一中对风险性异常判断的相关检验方法，为每一种方法附上不同的权重。

思路如图所示：

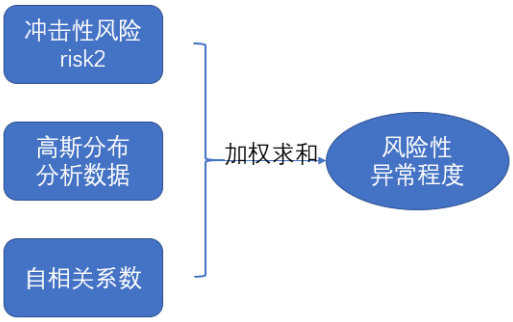

图 12: **问题二思路流程**

5.3.2 数据预处理

为了计算出每个传感器在不同时刻的异常性风险，我们需要结合问题一中第二步、第三步和第四步的相关算法。进行各传感器的相关计算并得出每个传感器在不同时刻的异常风险值。我们通过

问题一中步骤三的思想，针对某一时刻前后五个时刻的相关高斯分布情况进行了数据分析。通过问题一中步骤四的思想。分别计算了当 k=5 时，每一时刻与其之前所有时刻的自相关性系数的计算，并得出相应数据。将以上两部分数据分别以 csv 格式存入表格中。考虑到时间序列过于紧凑以及序列不可分、数据量大的性质，我们从对 0 点 23 点，我们以 30 分钟为单位进行了传感器总风险的计算。

5.3.3 模型建立

模型考虑到高斯分布情况及自相关性均可对风险性异常的持续性和联动性进行评估，因此，风险型异常数据异常程度的量化评价方法通过对以上两方面进行不同的权重赋值，得到最终的风险性异常结果。

考虑到自相关系数的可靠性更强，对持续性和联动性的评估结果更好，所以其权重会更大一些。通过多次实验，我们最终将 KS 检验结果与自相关性结果以 1:4 的权重进行计算，得到最终的异常结果分数。

异常结果存入 risk.csv 中。该表内部包含了时刻、异常性评价总分'total' 及各时刻对不同传感器的具体评分。针对每一行异常结果分数，我们增加了'Total' 列，来表征每个时刻的总异常得分。通过对该列进行降序排序，得到风险性最高的五个时刻，并分别对该五个时刻进行 100 个传感器异常得分的降序排序，取五个对该时刻异常值“贡献”最大的五个传感器编号。由于题目要求异常分数在 [0, 100]，因此对于所有异常分数乘以 0.35 进行处理。

最终得到的结果如下表所示：

	第一高分	**第二高分**	**第三高分**	**第四高分**	**第五高分**
异常程度得分	88.734	88.420	86.225	85.760	85.476
异常时刻编号	17:30:00	18:30:00	14:30:00	17:00:00	22:00:00
异常传感器编号	71	71	11	96	11
异常传感器编号	96	96	99	71	99
异常传感器编号	62	65	65	62	96
异常传感器编号	75	57	71	90	71
异常传感器编号	57	90	57	57	65

表 2: 问题 2 的结果

5.3.4 模型评价与推广

由于我们在方法一中的步骤三、四两部分内容选取参数问题（例如自相关性系数中 k 的取值），导致 risk.csv 风险表中还是会有部分错误产生。另外，由于各时刻数据时间间隔仅有 15 秒，如果全部考虑会导致风险表中数据过于密集，因此不得不对事件进行了筛选，考虑某一段时间间隔内的风险。虽然在这种情况下，由于自相关性在时间上的线性特性，可以保证时间间隔之间的数据点仍被算法考虑，但是这样仍会在一定程度上产生相应误差，对最终风险时刻产生影响。模型的进一步推广是尝试使用多个参数对数据进行处理并在不同的权重上进行评估，同时考虑总共 5519 个时刻的总体情况，让结果更加准确。

5.4 问题三分析

5.4.1 思路分析

问题三要求我们建立风险性异常预警模型，预测当日 23:00:00-23:59:59 可能产生的风险性异常。结合问题二中的风险性异常程度量化评价方法，考虑到前 23h 的风险性异常程度数据量过大且序列不可分，我们将前 23h 的风险异常程度数据的时间间隔放大至 15min。再分别从线性模型与非线性模型两个方面进行拟合，比较拟合预测效果，选择效果相对最好的模型作为我们的风险性异常预警模型。

5.4.2 数据预处理

我们将问题二中得到的前 23h 的风险量化程度数据的时间间隔缩小至 15min，确保拟合效率

5.4.3 模型建立

我们首先尝试了线性模型，拟合效果如图所示：

```
linear_regressor = linear_model.LinearRegression()
plt.plot(X, y, 'b.')
# 用特征构造数据进行预测
plt.plot(X, linear_regressor.fit(X,y).predict(X), 'r')
plt.show()
```

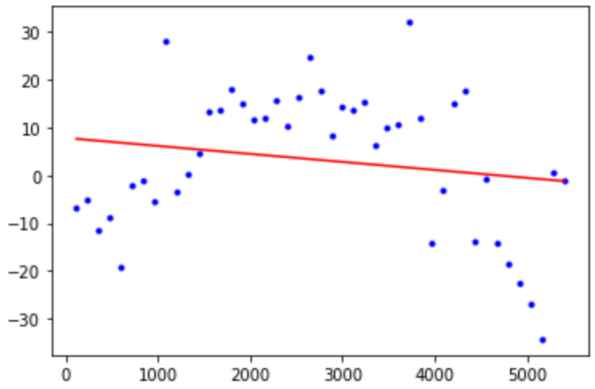

图 13: **线性模型拟合效果**

通过上图，很明显该模型欠拟合，存在大量游离的点。因此我们尝试了 RBF 核模型，拟合效果如图所示：

```
from sklearn.svm import SVR
# 拟合回归模型
svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
plt.plot(X, y, 'b.')
# 用特征构造数据进行预测
plt.plot(X, svr_rbf.fit(X,y).predict(X), 'r')
plt.show()
```

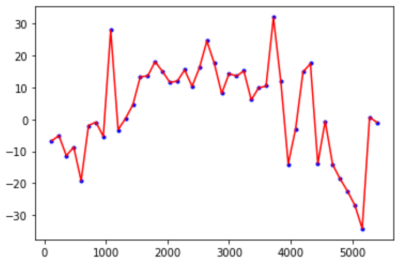

图 14: RBF **核模型拟合效果**

如上图所示，该模型存在过拟合的问题，预测时的泛化性较低。因此，我们又尝试了多项式非线性模型，拟合效果如图所示：

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures


# 特征构造
poly_reg = PolynomialFeatures(degree=2)
x_poly = poly_reg.fit_transform(X)
# 创建线性模型
linear_reg = LinearRegression()
linear_reg.fit(x_poly, y)
plt.plot(X, y, 'b.')
# 用特征构造数据进行预测
plt.plot(X, linear_reg.predict(poly_reg.fit_transform(X)), 'r')
plt.show()
```

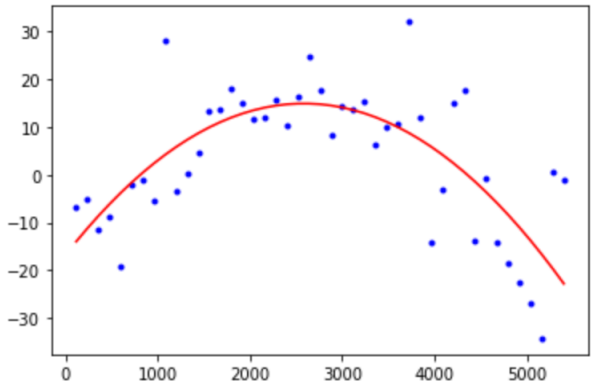

图 15: **多项式非线性拟合效果**

如图，该模型的拟合效果相对较好，能大致描述数据的特征，同时保证了一定的泛化能力。

经过不同模型的拟合预测比较，我们最终选择了多项式非线性模型作为我们的风险性异常预警模型。

最终预测结果如下 (异常分数与问题二进行了同样的处理)：

时间	23:00:00-23:14:59	23:15:00-23:29:59	23:30:00-23:44:59	23:45:00-23:59:59
异常最高分	75.778	74.848	73.806	72.764
异常传感器编号	54	54	54	54
异常传感器编号	20	20	20	20
异常传感器编号	33	33	33	33
异常传感器编号	2	2	2	5
异常传感器编号	47	47	47	2

表 3: 问题 3 的结果

5.4.4 模型评价与推广

为了确保拟合的效率与原时间序列的不可分割性，我们将前 23h 的采样间隔扩大到 15min。并尝试了多种模型的拟合效果，经过比较最终选择了多项式非线性模型，该模型较好地拟合了前 23h 的风险异常程度数据，并具备较好的泛化能力，做出的预测科学可靠，具有可信度。

5.5 问题四分析

5.5.1 思路分析

问题四要求建立数学模型，对该生产企业整个生产系统的安全性进行评价，即以 30 分钟为单位，如 00:30:00、01:00:00 等时刻，对生产系统给出一个最高分为 100、最低分为 0 的安全分数。对此，我们建立的数学模型首先套用了问题二的数学模型，来得到安全得分数据。对于同一时段不同传感器的安全得分，模型对其进行等权重相加求和，得到同一时段的总安全得分，Max-Min 标准化即离散标准化的方法，将得分值映射到 $[0,100]$ 之间，从而输出生产系统每一段时段的总安全分数。

5.5.2 数据预处理

我们将原数据以每 30 分钟的间隔，处理为各含 120 个时序数据的 46 个组，并对其进行安全得分计算，和问题三得到的预测安全得分合并，部分数据如图所示：

```
pd.set_option('display.max_rows', 10)
d_30.T
```

	1	2	3	4	5	6	7	8	9	10	...	91	92	
119	-6.875580	-6.363735	29.418405	9.414224	-11.856500	-27.839233	-50.000000	-50.000000	47.916667	7.944500	...	-12.742674	47.367674	-8.8276
239	-5.189924	36.920913	28.491223	8.264976	15.914608	-12.089677	-39.970796	-40.000000	40.021760	8.770087	...	-3.883858	5.731800	-33.6408
359	-11.480558	38.036705	36.923405	11.002403	22.809273	-15.859768	39.473649	39.444444	40.299538	7.168345	...	-10.470170	9.796657	-33.4371
479	-8.763139	39.280243	30.141360	-18.214190	16.955183	-8.252163	-39.970796	-40.000000	40.438427	10.317633	...	-18.376478	27.895642	-24.7464
599	-19.276822	40.059714	28.874149	7.650082	16.984718	-6.856926	39.695871	-40.000000	40.521760	7.826505	...	-6.380357	16.132886	-33.2207
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
5279	0.545920	39.856738	11.988856	33.400924	8.016352	-1.409422	39.767639	39.981320	38.192884	35.885663	...	19.868879	13.294725	10.5255
5399	-1.083898	39.854227	10.099433	17.044117	8.186292	-38.485525	39.742226	-40.000000	38.194408	34.921232	...	12.740241	14.726442	7.6554
5519	-24.006137	44.431400	11.478823	26.670984	29.716657	-12.912966	34.517674	-18.897690	34.152818	42.140048	...	18.069156	13.405387	-0.3296
5639	-26.882840	45.844066	10.896336	28.826057	38.514539	-13.951473	32.546413	-19.764152	32.892513	42.776115	...	19.352118	12.551699	-4.1481
5759	-29.839204	47.436401	10.283511	31.150352	48.265209	-15.133147	30.332901	-20.647258	31.482462	43.264762	...	20.820547	11.606411	-8.4846

48 rows × 100 columns

图 16: 30min **间隔与问题三中的预测安全得分合并后部分数据**

5.5.3 模型建立

我们首先套用问题二中已经建立好的安全分数评价方法，对输入的原始数据加预测数据进行安全评估，得到每 30 分钟的一个安全分数，之后，对于同时间段的不同传感器的安全得分进行等权重的相加求和，作为该时间段安全系统的总得分，部分得分情况如图所示：

```
pd.set_option('display.max_columns',10)
d_30t=d_30.T
d_30t['total']=d_30.describe().T['mean']*100
d_30t
```

	1	2	3	4	5	...	97	98	99	100	total
119	-6.875580	-6.363735	29.418405	9.414224	-11.856500	...	-7.462000	-50.000000	25.668797	47.916667	607.692976
239	-5.189924	36.920913	28.491223	8.264976	15.914608	...	-13.528687	-39.038852	23.721991	39.195618	683.611705
359	-11.480558	38.036705	36.923405	11.002403	22.809273	...	13.316478	40.405593	27.900323	14.991620	1488.318743
479	-8.763139	39.280243	30.141360	-18.214190	16.955183	...	-5.027509	-39.038852	28.782803	38.290155	769.689747
599	-19.276822	40.059714	28.874149	7.650082	16.984718	...	-29.475016	-39.038852	28.962431	38.408840	1216.135641
...	...	...	...	...	...	...	...	...	...	...	...
5279	0.545920	39.856738	11.988856	33.400924	8.016352	...	-28.396117	30.903334	40.055709	41.424904	2442.181008
5399	-1.083898	39.854227	10.099433	17.044117	8.186292	...	-31.941227	30.907193	40.064524	41.424899	2265.973663
5519	-24.006137	44.431400	11.478823	26.670984	29.716657	...	-21.910520	30.218787	39.492137	41.590749	2191.637637
5639	-26.882840	45.844066	10.896336	28.826057	38.514539	...	-21.480389	30.274795	39.383039	41.620086	2138.521155
5759	-29.839204	47.436401	10.283511	31.150352	48.265209	...	-20.805039	30.397422	39.277245	41.642947	2078.961224

48 rows × 101 columns

图 17: **部分安全系统总得分情况**

得到总得分后，按照题目要求需要将其映射到 $[0,100]$ 之间，故对上一步得到的总得分进行 Max-Min 标准化，公式为：

$$new\_score = 100 * \frac{score - min\_score}{max\_score - min\_score} \tag{17}$$

最终得到结果，部分数据如图所示：

```
d_30t['safe_score']=finals
d_30t
```

	1	2	3	4	5	...	98	99	100	total	safe_score
119	-6.875580	-6.363735	29.418405	9.414224	-11.856500	...	-50.000000	25.668797	47.916667	607.692976	100.000000
239	-5.189924	36.920913	28.491223	8.264976	15.914608	...	-39.038852	23.721991	39.195618	683.611705	96.061427
359	-11.480558	38.036705	36.923405	11.002403	22.809273	...	40.405593	27.900323	14.991620	1488.318743	54.314188
479	-8.763139	39.280243	30.141360	-18.214190	16.955183	...	-39.038852	28.782803	38.290155	769.689747	91.595801
599	-19.276822	40.059714	28.874149	7.650082	16.984718	...	-39.038852	28.962431	38.408840	1216.135641	68.434722
...	...	...	...	...	...	...	...	...	...	...	...
5279	0.545920	39.856738	11.988856	33.400924	8.016352	...	30.903334	40.055709	41.424904	2442.181008	4.828954
5399	-1.083898	39.854227	10.099433	17.044117	8.186292	...	30.907193	40.064524	41.424899	2265.973663	13.970381
5519	-24.006137	44.431400	11.478823	26.670984	29.716657	...	30.218787	39.492137	41.590749	2191.637637	17.826845
5639	-26.882840	45.844066	10.896336	28.826057	38.514539	...	30.274795	39.383039	41.620086	2138.521155	20.582464
5759	-29.839204	47.436401	10.283511	31.150352	48.265209	...	30.397422	39.277245	41.642947	2078.961224	23.672362

48 rows × 102 columns

图 18: **最终结果部分数据**

最终数据为：

时间	安全性评分	时间	安全性评分
00:30:00	100	12:30:00	18.59041661
01:00:00	96.06142712	13:00:00	29.42149381
01:30:00	54.31418813	13:30:00	5.49713625
02:00:00	91.59580123	14:00:00	20.06928933
02:30:00	68.43472201	14:30:00	3.71910903
03:00:00	37.60954555	15:00:00	16.0344047
03:30:00	75.28877471	15:30:00	10.52945203
04:00:00	80.71532653	16:00:00	5.27700308
04:30:00	89.67610849	16:30:00	10.56942912
05:00:00	53.40050586	17:00:00	4.40912593
05:30:00	56.65196723	17:30:00	0
06:00:00	25.16781823	18:00:00	7.50488318
06:30:00	59.19368574	18:30:00	0.46438034
07:00:00	63.03535839	19:00:00	14.73537951
07:30:00	50.27645299	19:30:00	11.31711514
08:00:00	54.28388029	20:00:00	13.16546178
08:30:00	40.29382475	20:30:00	17.1589822
09:00:00	53.08524785	21:00:00	5.56141562
09:30:00	37.94986815	21:30:00	15.9821449
10:00:00	28.92019905	22:00:00	4.82895433
10:30:00	21.05268287	22:30:00	13.9703807
11:00:00	29.2886668	23:00:00	17.82684484
11:30:00	17.97606041	23:30:00	20.58246438
12:00:00	18.54299488	24:00:00	23.67236239

表 4: 问题 3 的结果

5.5.4 **模型评价与推广**

根据模型结论，该安全系统当日安全得分情况如下图所示：

```
plt.scatter(al, finals)
plt.show()
```

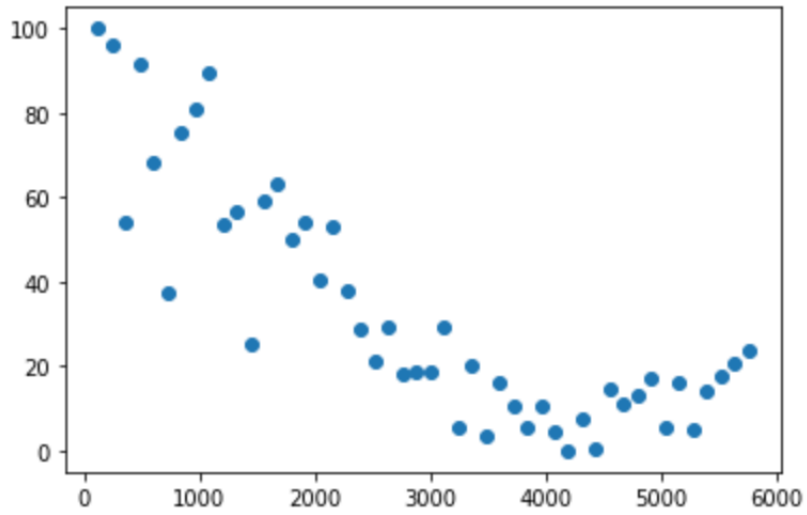

图 19: **该安全系统当日得分情况**

由图可以看出，当天安全情况在一天的最开始时最好，为 100 分，在一天的运行中由于各种环境情况，安全系数呈下降趋势，与常理相符，可以认为评价正确。

6 参考文献

[1] **周志华**. **机器学习** [M]. **北京**：**清华大学出版社**，2016：225-266

[2]https://blog.csdn.net/qq$_4$1679006/*article*/*details*/80977113

[3]*https* : //*jacobzhuo.blog.csdn.net*/*article*/*details*/117534793

[4]*https* : //*blog.csdn.net*/*zzminer*/*article*/*details*/8858469

[5]*https* : //*pandas.pydata.org*/*pandas*−*docs*/*stable*/*reference*/*api*/*pandas.Series.autocorr.html*

[6]*https* : //*github.com*/*scipy*/*scipy*/*blob*/*v*1.4.1/*scipy*/*stats*/*stats.py*

[7]*Y.Suetal.*,"*CoFlux* : *RobustlyCorrelatingKPIsbyFluctuationsforServiceTroubleshooting*, "2019*IEEE*/*ACM*27*thInternationalSymposiumonQualityofService*(*IWQoS*), 2019, *pp*.1−10, *doi* : 10.1145/3326285.3329048.

[8]*https* : //*blog.csdn.net*/*weixin*$_4$3398590/*article*/*details*/106307466

[9]*https* : //*blog.csdn.net*/*kangjielearning*/*article*/*details*/104686096

[10]*https* : //*blog.csdn.net*/*qushoushi*0594/*article*/*details*/80096213

[11]*https* : //*zhuanlan.zhihu.com*/*p*/71178532

附录

A 附件 1 相关程序

问题一模型

```
#matplotlib inline
from copy import deepcopy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn import preprocessing
import xlrd
import os
from pandas import DataFrame
from pandas import Series
print("导入成功")

plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')



# 导入数据集
data = pd.read_excel('C:\\Users\\Lenovo\\Desktop\\培训题1\\
附件1.xlsx')
#后期去除了完全平滑的情况，8，35，48
newdata=pd.read_excel('C:\\Users\\Lenovo\\Desktop\\培训题1\\
new附件1.xlsx')
print(data.shape)
data.head()

df=data
def fun(df):
    dic = {}
    for i in df.columns:
        dic[i] = df[i].value_counts()
    return dic

diff = fun(df)
```

```
diff[7]

countDiff=[]
for i in range(1,101):
    countDiff.append(len(dd[i]))
    print(i,":",len(dd[i]))
print(countDiff)

import cv2
# encoding=utf-8
from pylab import *                                    #支持中文
mpl.rcParams['font.sans-serif'] = ['SimHei']

x=range(0,5519)
fig = plt.figure(figsize=(15,50))
fig.suptitle("每个传感器数据的折线图",fontsize=30)



for i in range(2,102):
    y=data.iloc[:,i]
    ax = plt.subplot(25,4,i-1)
    plt.subplots_adjust(hspace=0.5)
    #plt.plot(x, y, marker='o', mec='r', mfc='w')#,label
    =u'y=x^2曲线图')
    plt.plot(x,y,linewidth=0.5,c='royalblue')
    #fig, axes = plt.subplots(1, 1, figsize=(8, 4))

    # 折线图, color='#DE6B58'
    #axes.plot(x, y, linestyle='-', marker='x', linewidth=1000)

    #plt.legend()  # 让图例生效
    #plt.xticks(x, names, rotation=45)
    #plt.margins(0)
    #plt.subplots_adjust(bottom=0.15)

    plt.xlabel(f"第"+str(i-1)+"个")#"时间序列计数") #X轴标签
    #plt.ylabel("取值") #Y轴标签
```

```
    #plt.title(" #标题
#plt.show()

# -*- coding: utf-8 -*-
def location_corresponding(x_str, s_str):
    '''
    s 的第三位有效数字，以及对应 x 的相应的数字
    '''
    digit_location = s_str.find('.')
    if digit_location >= 3:
        # 如果小数点在字符串的第 4 位或大于第四位，也即数字至少
        在 100 以上
        # 此时第三位有效数字，肯定在小数点前。
        significant_num = s_str[2]
        # 有效位数于小数点的相对位置
        # 0 代表在小数点前， 3-1 是指从小数点前数起的位数
        # sig_loc 的第一位表示有效数字在小数点前，还是后
        # 第二位代表有效数字在小数点的 “距离”
        str_len = len(s_str[:digit_location])
        sig_loc = (0, str_len-3-1)

    elif s_str[0] == '0':
        # 若整数部分是0，则有效数字的位置在小数点后
        # 刨除整数和小数点部分
        s_without_int = s_str[digit_location:]
        # 若小数点后有 0，则不将 0 计入有效数字位
        if '.0' in s_without_int:
            digit_location += 1
            while '.00' in s_without_int:
                # 若小数点后有多个0，也不计入
                s_without_int = s_without_int.replace('.00',
                '.0')
                digit_location += 1
        try:
            # 若位数不够，如 0.0，则有效数字为 0、
            significant_num = s_str[digit_location+3]
        except:
```

```
            significant_num = '0'
        # 1 表示有效数字在小数点后
        sig_loc = (1, 3)

    elif digit_location == -1 and len(s_str) >= 3:
        # 若只有整数，没有小数部分，且整数部分大于 100，即有
        超过三位数
        # 则直接取第三位
        str_len = len(s_str)
        sig_loc = (0, str_len-3)
        significant_num = s_str[2]
    elif digit_location == -1 and len(s_str) < 3:
        # 若小于 100，则有效数字为0
        significant_num = '0'
    else:
        # 若整数部分小于 2 位，且整数部分大于 0
        sig_loc = (1, 3-digit_location)
        significant_num = s_str[sig_loc[1]+digit_location]

    # x 对应的数字
    x_digit_location = x_str.find('.')
    if sig_loc[0] == 0:
        if x_digit_location == -1 and len(x_str) >= 3:
            x_significant_num = x_str[2]
        elif x_digit_location == -1 and len(x_str) < 3:
            x_significant_num = '0'
        else:
            x_significant_num = x_str[x_digit_location-sig_loc[1]]
            print(sig_loc[1])
            print('前')
    else:
        try:
            x_significant_num = x_str[x_digit_location+sig_loc[1]]
        except:
            x_significant_num = '0'

    return x_significant_num, significant_num
```

```
def coverage_critiria(x_list, s_list):
    '''
    收敛准则
    其中 s_list 是一个长度为 3 的 list，包含当前迭代的 s* 和
    之前两个迭代的 s*
    其中 x_list 也一样

    难点在于：如何找出 s 的第三位有效数字，对应 x 的数位呢？
            这里的解决办法是：找出 s 三位有效数字，在小数点
            的位置，从而应用于 x 中
    '''


    s_numbers = []
    x_numbers = []
    for i in range(3):
        # 连续两位不变，故需要进行 3 此迭代。
        s = s_list[i]
        # s 的字符串
        s_str = str(s)

        x = x_list[i]
        # x 的字符串
        x_str = str(x)
        # 找出 s* 的第三位有效数字，和 x* 对应的数字。
        x_sig_num, s_sig_num = location_corresponding(x_str, s_str)
        s_numbers.append(s_sig_num)
        x_numbers.append(x_sig_num)

    # 稳健标准差的第三位有效数字，连续两次不变，且稳健平均值的
    对应数字亦连续两次不变，则判断为收敛
    # 函数返回 True。否则返回 False
    s_equal_flag = (s_numbers[0] == s_numbers[1]) and
    (s_numbers[0] == s_numbers[2])
    x_equal_flag = (x_numbers[0] == x_numbers[1]) \
                        and (x_numbers[0] == x_numbers[2])
```

```
    if s_equal_flag and x_equal_flag:
        return True
    else:
        return False

def perform_algorithm_A(x):
    '''
    x 是一个向量
    '''
    if type(x) is list:
        x = np.array(x)

    # 稳健平均和稳健标准差初始值
    x_star = np.median(x)
    x_diff = np.absolute(x-x_star)
    s_star = 1.483*np.median(x_diff)

    # 上界和下界的初始值
    delta = 1.5*s_star
    higher_bound = x_star + delta
    lower_bound =  x_star - delta

    x_list = []
    s_list = []
    while True:
        x_tmp = np.copy(x)
        x_tmp[x_tmp>higher_bound] = higher_bound
        x_tmp[x_tmp<lower_bound] = lower_bound
        x_star = np.mean(x_tmp)
        s_star = 1.134*np.std(x_tmp)

        x_list.append(x_star)
        s_list.append(s_star)

        if len(x_list) == 4:
            x_list.pop(0)
            s_list.pop(0)
```

```
            # 达到收敛条件
            if coverage_critiria(x_list, s_list):
                return x_tmp, x_list[2], s_list[2], higher_bound,
                lower_bound


        # 计算上下界
        delta = 1.5*s_star
        higher_bound = x_star + delta
        lower_bound =  x_star - delta

x_star_A=[]
s_star_A=[]
A_risk=[[]for i in range(100)]

#x中取值为上下界的个数几位m，冲击性风险=m/n
for i in range(2,102):
    x=data.iloc[:,i]
    x, xstar, sstar, higherbound, lowerbound = perform_algorithm_A(x)
    x_star_A.append(xstar)
    s_star_A.append(sstar)
    #print(i,": x_star=",xstar,"s_star=",sstar)
    n=len(x)
    m=0
    for j in range(n):
        if x[j]==higherbound or x[j]==lowerbound:
            m+=1
    risk_temp=m/n
    A_risk[i-2].append(i-1)
    A_risk[i-2].append(risk_temp)
    print(A_risk[i-2])

#x=[5,5,2,1,0,0,4,0,1,1]
#print(type(x))
def cal_risk(x,higherbound,lowerbound):
    r=0
    '''for i in range(10):
```

```
            n=len(x)
            m=0'''
        n=len(x)
        m=0
        temp=0
        for j in range(n):
            if temp<2:
                temp=0
                total=0
                while j<n:
                    #print(x[j])
                    if x[j]!=higherbound and x[j]!=lowerbound:
                        break
                    elif x[j]==higherbound or x[j]==lowerbound:
                        temp+=1
                    j+=1
                if temp!=0:
                    #print(temp)
                    total=(1+temp)*temp/2
                r+=total
                #print("total:",total)
            elif temp>=2:
                temp-=1
        #print("r:",r)
        return r

#print(cal_risk(x,5,0))

#带权重的A_algorithm
A_r=[]
for i in range(2,102):
    x=data.iloc[:,i]
    x, xstar, sstar, higherbound, lowerbound = perform_algorithm_A(x)
    x=x.tolist()
    m=cal_risk(x,higherbound,lowerbound)
    n=len(x)
    A_r.append(m/n)
```

```
    print(i-1,":",m/n)
#print(A_r)

record_A=[]
for i in range(100):
    if A_r[i]>=90 or A_r[i]==0:
        record_A.append(i+1)
print(record_A)

def pearson_corrlation(object1, object2):
    values = range(len(object1))

    # Summation over all attributes for both objects
    sum_object1 = sum([float(object1[i]) for i in values])
    sum_object2 = sum([float(object2[i]) for i in values])

    # Sum the squares
    square_sum1 = sum([pow(object1[i],2) for i in values])
    square_sum2 = sum([pow(object2[i],2) for i in values])

    # Add up the products
    product = sum([object1[i]*object2[i] for i in values])

    #Calculate Pearson Correlation score
    numerator = product - (sum_object1*sum_object2/len(object1))
    denominator = ((square_sum1 - pow(sum_object1,2)/
    len(object1)) * (square_sum2 -
        pow(sum_object2,2)/len(object1))) ** 0.5

    # Can"t have division by 0
    if denominator == 0:
        return 0

    result = numerator/denominator
    return result

#风险信号的联动性和持续性，通过自相关系数进行判断
```

```
#自相关系数：当前时刻和其前k个时刻的Pearson相关系数计算公式
def get_auto_corr(timeSeries,k):
    '''
    Descr:输入：时间序列timeSeries，滞后阶数k
          输出：时间序列timeSeries的k阶自相关系数
        l：序列timeSeries的长度
        timeSeries1，timeSeries2:拆分序列1，拆分序列2
        timeSeries_mean:序列timeSeries的均值
        timeSeries_var:序列timeSeries的每一项减去均值的平方的和
    '''
    l = len(timeSeries)
    #取出要计算的两个数组
    timeSeries1 = timeSeries[0:l-k]
    timeSeries2 = timeSeries[k:]
    timeSeries_mean = timeSeries.mean()
    timeSeries_var = np.array([i**2 for i in timeSeries
    -timeSeries_mean]).sum()
    auto_corr = 0
    for i in range(l-k):
        temp = (timeSeries1[i]-timeSeries_mean)*(timeSeries2[i]
        -timeSeries_mean)/timeSeries_var
        auto_corr = auto_corr + temp
    return auto_corr

def pearson(current_num,sensor,k):
    x = data.iloc[:,sensor+1]#选中相关传感器的那列
    #print(type(x))
    timeSeries = x[0:current_num]#选中该时刻及之前的时间序列
    #print(timeSeries)
    timeSeries=timeSeries.values
    corr = get_auto_corr(timeSeries,k)
    return corr
'''current_num可以从0开始，sensor只能是1-100'''

#print(pearson(5519,7,8))
selfcorr_list_8=[]
selfcorr_list_4=[]
```

```
for i in range(100):
    temp1=pearson(5519,i+1,8)
    selfcorr_list_8.append(temp1)
    temp2=pearson(5519,i+1,4)
    selfcorr_list_4.append(temp2)
print(selfcorr_list_8)

selfcorr_df_8=pd.DataFrame(selfcorr_list_8,columns=['corrlation'])
selfcorr_df_4=pd.DataFrame(selfcorr_list_4,columns=['corrlation'])
#selfcorr_df_8.iloc[0,1]='1'
#selfcorr_df_4.iloc[0,1]='1'
#print(selfcorr_df_8)
for i in range(100):
    selfcorr_df_8.loc[i,'sensor_num']=str(i+1)
    selfcorr_df_4.loc[i,'sensor_num']=str(i+1)
    #去除平滑情况
    '''if i+1==8 or i+1==35 or i+1==48:
        selfcorr_df_8.drop(str(i),axis=0)
        selfcorr_df_4.drop(str(i),axis=0)'''
#print(selfcorr_df_8)
selfcorr_df_8=selfcorr_df_8.sort_values(by=['corrlation',
'sensor_num'],ascending=[False,True])
selfcorr_df_4=selfcorr_df_4.sort_values(by=['corrlation',
'sensor_num'],ascending=[False,True])

print("k=8:\n",selfcorr_df_8)
print("\nk=4:\n",selfcorr_df_4)

#画各阶自相关系数的图
import cv2
# encoding=utf-8
from pylab import *
def plot_auto_corr(t,timeSeries,k):
    '''
    Descr:需要计算自相关函数get_auto_corr(timeSeries,k)
          输入时间序列timeSeries和想绘制的阶数k，k不能超过
          timeSeries的长度
```

```
        输出：k阶自相关系数图，用于判断平稳性
    '''
    #支持中文
    mpl.rcParams['font.sans-serif'] = ['SimHei']
    timeSeriestimeSeries = pd.DataFrame(range(k))
    for i in range(1,k+1):
        timeSeriestimeSeries.loc[i-1] =get_auto_corr(timeSeries,i)
    plt.bar(range(1,len(timeSeriestimeSeries)+1),
    timeSeriestimeSeries[0])
    plt.title("传感器"+str(t)+"在k="+str(k)+"时的自相关关系",
    fontsize=30)
    plt.ylabel("自相关系数取值",fontsize=20)
    return timeSeriestimeSeries

sensor=17 #sensor定位（1-100)
x = data.iloc[:,sensor+1]#选中实际输入-1的那列传感器数据，
e.g.输入48，选中的是第47个传感器
#print(x)
timeS = x[0:5519]#选中该时刻及之前的时间序列
#print(timeSeries)
timeS=timeS.values
print("k = 8:\n")
plot_auto_corr(sensor,timeS,13)
```

问题一模型（KS 检验）

```
#!/usr/bin/env python
# coding: utf-8

## 问题一步骤三
#
## [0. 读取文件](#0)
## [1. AD检验](#1)
## [2. KS检验](#2)

## <span id="0">0. 读取文件</span>

# In[1]:
```

```
import pandas as pd
import math
pi=math.pi
e=math.e

# In[3]:

data=pd.read_excel('附件1.xlsx')

# In[4]:


data=data.rename(columns={'时刻编号\n(Time number)':'tNumber','
传感器编号\n                                    (Sensor number)\n
时刻 (Time)':'time'})
data

## <span id="1">1. AD检验<\span>
#
#
# y=x-meanx/std
# fy=(1/sqrt(2pi))*e^(-y^2/2)

# In[59]:

for i in range(1,100):
    if stodata[i]['min']==stodata[i]['max']:
        data.drop(columns=[i])

data


# In[6]:

stodata=data.describe()
```

```
# In[26]:

y_data=pd.DataFrame()
y_data['tNumber']=data['tNumber']

for i in range(1,100):
    y_data[i]=(data[i]-stodata[i]['mean'])/stodata[i]['std']

y_data

# In[16]:



A2list=[]
ADlist=[]

for i in range(1,101):
    i=str(i)
    d=pd.DataFrame()
    d[i]=(data[i]-stodata[i]['mean'])/stodata[i]['std']
    d.sort_values(by=i,inplace=True)
    #d['f']=e**(-(d[i])**2/2)/(stodata[i]['std']*(2*pi)**0.5)
    d['f']=st.norm.cdf(d[i])
    if eval(i) in outlist:
        continue
    sumd=0
    num=d.shape[0]
    for j in range(num):
        sumd=sumd+(2*j+1)*math.log(d['f'][j])+(2*num-2*j-1)*
        math.log(1-d['f'][j])
    print(sumd)
    A2=(-num-sumd/num)/num
    AD=A2*(1+0.75/num+2.25/(num**2))
    print(d)
    print(A2)
    print(AD)
    A2list.append(A2)
```

```
    ADlist.append(AD)

# In[11]:

import scipy.stats as st
st.norm.cdf(0.057053)

# In[71]:


d=pd.DataFrame()
d[i]=(data[i]-stodata[i]['mean'])/stodata[i]['std']
d


# In[48]:


d=pd.DataFrame()
d[1]=(data[1]-stodata[1]['mean'])/stodata[1]['std']
d.sort_values(by=1,inplace=True)
d=d.reset_index(drop=True)
d['f']=e**(-(d[1])**2/2)/(stodata[1]['std']*(2*pi)**0.5)#*10000
d

## <span id='2'>2. KS检验<\span>
#

# In[84]:


import numpy as np
from scipy.stats import  kstest
P_ks = []#  这个是KS检验的P值
pks2 = []
pks3=[]
d=pd.DataFrame()
```

```
for i in range(1,101):
    i=str(i)
    #data_t = [j for j in data[i] if  np.percentile(data[i],
    (25,75))[0] <j< np.percentile(data[i],(25,75))[1] ]
    #if data_t :
    #    data_i = data_t
    #else :
    #    data_i = data[i]

    d[i]=(data[i]-stodata[i]['mean'])/stodata[i]['std']
    #d.sort_values(by=i,inplace=True)

    teststat, pvalue = kstest(rvs=d[i],cdf='norm',N=5519,
    mode = 'asymp')
    teststat, pvalue2 = st.shapiro(d[i])
    teststat, pvalue3 = st.normaltest(d[i])
    #print((teststat,pvalue))
    P_ks.append(pvalue)
    pks2.append(pvalue2)
    pks3.append(pvalue3)
#P_ks = np.array(P_ks)
print(P_ks,pks2,pks3)


# In[86]:


math.isnan(d['8'][0])


# In[99]:


d.info(verbose=True,null_counts=True)


# In[87]:
```

```
rvs=d[4990:4995]['100']
type(rvs)


# In[93]:


s[5517:5520].shape[0]


# In[88]:


s=pd.DataFrame()
s[i]=[0]*5519
s


# In[114]:


s=pd.DataFrame()
for i in range(1,101):
    i=str(i)
    s[i]=[0]*5519
    if i=='8' or i=='35':
        continue
    for j in range(0,92):#(0,5519):
        #if j>=2:
        #    rvs=d[j-2:j+3][i]
        #else:
        #    rvs=d[:j+3][i]
        rvs=d[(60*j):(60*j+60)][i]
        vals = np.sort(rvs)
        N = len(vals)
```

```
        cdfvals = st.norm.cdf(vals)
        Dplus = (np.arange(1.0, N + 1)/N - cdfvals).max()
        Dmin = (cdfvals - np.arange(0.0, N)/N).max()
        D = np.max([Dplus, Dmin])
        p_value=st.kstwobign.sf(D * np.sqrt(N))
        s.loc[j,i]=p_value


# In[120]:


al=[]
for i in range(1,93):
    al.append(60*i-1)
al


# In[118]:


s2=s[:46]


# In[119]:


s2.to_csv('score3_30min.csv',index=None)


# In[68]:


from torch import distributions
import numpy as np

#cdf='norm'
rvs=d[4990:4995]['100']
```

```
#for j in range(5519):
#    cdf = getattr(distributions, cdf).cdf
vals = np.sort(rvs)
N = len(vals)
cdfvals = st.norm.cdf(vals)
Dplus = (np.arange(1.0, N + 1)/N - cdfvals).max()
Dmin = (cdfvals - np.arange(0.0, N)/N).max()
D = np.max([Dplus, Dmin])
statics,p_value=D, st.kstwobign.sf(D * np.sqrt(N))


# In[69]:


p_value


# In[185]:


adf=Out[174]
adf['ketest++']=np.array(P_ks)
adf


# In[81]:


pd.set_option('display.max_rows', 10)
bdf[bdf['kstest']==0].index


# In[124]:


pd.set_option('display.max_rows', 30)
bdf=pd.DataFrame({'kstest':P_ks,'shapiro':pks2,
```

```
'normaltest':pks3})


# In[127]:


bdf


# In[125]:


bdf[bdf['normaltest']==0]


# In[126]:


bdf[bdf['kstest']==0]
```

问题二模型

```
#第二问
Q2_3=Q2_2#5519*101
#print(Q2_3)
for i in range(100):
    for j in range(5519):
        temp=pearson(j+1,i+1,5)
        #5519*100，定位到第j行第i列，修改数字
        Q2_3.loc[j,str(i+1)]=temp
#index=False,header=False表示不保存行索引和列标题
save.to_csv('C:\\Users\\Lenovo\\Desktop\\培训题1\\score3.csv',
index=False,header=False)

#以30min为周期，行数为以下
time_rownum=[119,239,359,479,599,719,839,959,1079,1199,1319,1439,
          1559,1679,1799,1919,2039,2159,2279,2399,2519,2639,2759,
         2879,2999,3119,3239,3359,3479,3599,3719,3839,3959,4079,
```

```
            4199,4319,4439,4559,4679,4799,4919,5039,5159,5279,5399]
#time_row=['00:30:00',]

Q2_risk = pd.read_csv('C:\\Users\\Lenovo\\Desktop\\培训题1\\Q2_risk.csv')
Q2_3_30=Q2_risk
Q2_2_30=Q2_risk
for i in range(len(time_rownum)):
    m=time_rownum[i]
    temp2=Q2_2.loc[m]
    Q2_2_30.loc[Q2_2_30.index.max()+1]=temp2
    temp3=Q2_3.loc[m]
    Q2_3_30.loc[Q2_3_30.index.max()+1]=temp3
    #2
    Q2_2_30=Q2_2_30.reset_index()
    Q2_2_30.drop(columns=['index'],inplace=True)
    #3
    Q2_3_30=Q2_3_30.reset_index()
    Q2_3_30.drop(columns=['index'],inplace=True)

#加权重
#Q2.loc[1,100]=Q2_2_30.loc[1,100]+Q2_3_30.loc[1,100]
#Q2.loc[0]=Q2_2_30.loc[0]
Q2=10*Q2_2_30+40*Q2_3_30
Q2['Time']=Q2_2_30['Time']
print(20*Q2_3_30)#45*101

#计算每个时刻之和
Q2['total']=''
for i in range(45):
    temp=0
    for j in range(1,101):
        temp+=Q2.loc[i,str(j)]
    Q2.loc[i,'total']=temp
print(Q2)

#for i in range(100):
#    Q2.loc[i,'sensor_num']=str(i+1)
```

```
    #去除平滑情况
#     '''if i+1==8 or i+1==35 or i+1==48:
#         selfcorr_df_8.drop(str(i),axis=0)
#         selfcorr_df_4.drop(str(i),axis=0)'''
#print(selfcorr_df_8)
Q2=Q2.sort_values(by=['total','Time'],ascending=[False,True])

print("第二问风险时刻排序:\n",Q2)

a=pd.DataFrame(a,columns=['risk'])
#print(a)
#selfcorr_df_8.iloc[0,1]='1'
#selfcorr_df_4.iloc[0,1]='1'
#print(selfcorr_df_8)
for i in range(100):
    a.loc[i,'sensor_num']=str(i+1)
    #去除平滑情况
    '''if i+1==8 or i+1==35 or i+1==48:
        selfcorr_df_8.drop(str(i),axis=0)
        selfcorr_df_4.drop(str(i),axis=0)'''
#print(selfcorr_df_8)
a=a.sort_values(by=['risk','sensor_num'],ascending=[False,True])
print("Risk:\n",a)

Q2_origin=Q2
print(type(Q2))
Q2_origin.to_csv('C:\\Users\\Lenovo\\Desktop\\培训题1\\risk.csv')
#Q2=pd.read_csv('C:\\Users\\Lenovo\\Desktop\\培训题1\\risk.csv')
print(Q2)
```

问题三模型

```
#!/usr/bin/env python
# coding: utf-8

## 问题三
#
## [1. 读入数据](#1)
```

```
## [2. 处理数据](#2)
## [3. 线性回归预测](#3)

## <span id='1'>1. 读入数据</span>

# In[1]:


from sklearn import datasets
import matplotlib.pyplot as plt


# In[2]:


from sklearn import linear_model


# In[3]:


import pandas as pd


# In[4]:


import math


# In[119]:


seq=pd.read_csv('data.csv').loc[:,['tNumber','time']]
seq
```

```
# In[5]:


data2=pd.read_csv('risk1.csv')
data2


# In[11]:


data=pd.read_csv('score_z.csv')
data


## <span id='2'>2. 处理数据</span>

# In[41]:


al=[]
for i in range(1,93):
    al.append(60*i-1)
al


# In[16]:


data['7'][50:60]


# In[29]:


d#=data.loc[al]
```

```
# In[14]:


data2.info(verbose=True,null_counts=True)


# In[33]:


d.info(verbose=True,null_counts=True)


# In[12]:


needfill=[7,8,9,13,20,22,24,27,28,29,35,40,41,42,43,44,45,46,47,48,50,
          52,55,58,67,76,77,80,83,90,98,100]


# In[26]:


i='7'
j=59
if math.isnan(d[i][j]):#空值，需要填充
    m=1
    while(math.isnan(data[i][j-m])):
        m+=1
    d.loc[j,i]=data[i][j-m]
d[i][j]


# In[32]:


d#d.loc[i,j]#=data[i][j-m]
```

```
# In[31]:


for i in needfill:
    i=str(i)
    for j in al:
        if math.isnan(d[i][j]):#空值，需要填充
            m=1
            while(math.isnan(data[i][j-m])):
                m+=1
            d.loc[j,i]=data[i][j-m]


## <span id='3'>3. 线性回归预测</span>
#
## 处理完毕！开始预测

## 核方法

# In[ ]:


from sklearn.svm import SVR
# 拟合回归模型
svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
svr_lin = SVR(kernel='linear', C=100, gamma='auto')
svr_poly = SVR(kernel='poly', C=100, gamma='auto', degree=3, epsilon=.1,
               coef0=1)

################################################################################
# 查看结果
lw = 2

svrs = [svr_rbf, svr_lin, svr_poly]
kernel_label = ['RBF', 'Linear', 'Polynomial']
model_color = ['m', 'c', 'g']
```

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 10), sharey=True)
for ix, svr in enumerate(svrs):
    axes[ix].plot(X, svr.fit(X, y).predict(X), color=model_color[ix], lw=
                  label='{} model'.format(kernel_label[ix]))
    axes[ix].scatter(X[svr.support_], y[svr.support_], facecolor="none",
                     edgecolor=model_color[ix], s=50,
                     label='{} support vectors'.format(kernel_label[ix]))
    axes[ix].scatter(X[np.setdiff1d(np.arange(len(X)), svr.support_)],
                     y[np.setdiff1d(np.arange(len(X)), svr.support_)],
                     facecolor="none", edgecolor="k", s=50,
                     label='other training data')
    axes[ix].legend(loc='upper center', bbox_to_anchor=(0.5, 1.1),
                    ncol=1, fancybox=True, shadow=True)

fig.text(0.5, 0.04, 'data', ha='center', va='center')
fig.text(0.06, 0.5, 'target', ha='center', va='center', rotation='vertica
fig.suptitle("Support Vector Regression", fontsize=14)
plt.show()


# In[23]:


from sklearn.svm import SVR
# 拟合回归模型
svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
svr_lin = SVR(kernel='linear', C=100, gamma='auto')
svr_poly = SVR(kernel='poly', C=100, gamma='auto', degree=2, epsilon=.1,
               coef0=1)
svrs = [svr_rbf, svr_lin, svr_poly]
#for ix, svr in enumerate(svrs):
#    svr.fit(X,y)


# In[36]:
```

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures


# 特征构造
poly_reg = PolynomialFeatures(degree=3)
x_poly = poly_reg.fit_transform(X)
# 创建线性模型
linear_reg = LinearRegression()
linear_reg.fit(x_poly, y)
plt.plot(X, y, 'b.')
# 用特征构造数据进行预测
plt.plot(X, linear_reg.predict(poly_reg.fit_transform(X)), 'r')
plt.show()


# In[38]:


X_train=np.array(a1[:45])
X_train=X_train.reshape(-1, 1)
X_test=np.array([5519,5639,5759])
X_test=X_test.reshape(-1, 1)
d2_pre=pd.DataFrame()
for i in range(1,101):
    i=str(i)
    y_train=list(data2[i].values)
    # 特征构造
    poly_reg = PolynomialFeatures(degree=3)
    x_poly = poly_reg.fit_transform(X_train)
    # 创建线性模型
    linear_reg = LinearRegression()
    linear_reg.fit(x_poly, y_train)
    #预测
    d2_pre[i] = linear_reg.predict(poly_reg.fit_transform(X_test))
d2_pre
```

```
# In[43]:


[5579,5639,5699,5759]


# In[46]:


d2_for=pd.DataFrame()
d2_for[5579]=(d2_pre.T[0]+d2_pre.T[1])/2
d2_for[5639]=d2_pre.T[1]
d2_for[5699]=(d2_pre.T[2]+d2_pre.T[1])/2
d2_for[5759]=d2_pre.T[2]
d2_for


# In[58]:


d2_for.sort_values(by=5759)


# In[53]:


d2_for.describe().T['mean']*100


# In[47]:


d2_for.T
```

```
# In[35]:


from sklearn.svm import SVR
# 拟合回归模型
svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
plt.plot(X, y, 'b.')
# 用特征构造数据进行预测
plt.plot(X, svr_rbf.fit(X,y).predict(X), 'r')
plt.show()


# In[34]:


linear_regressor = linear_model.LinearRegression()
plt.plot(X, y, 'b.')
# 用特征构造数据进行预测
plt.plot(X, linear_regressor.fit(X,y).predict(X), 'r')
plt.show()


# In[ ]:


linear_regressor = linear_model.LinearRegression()
linear_regressor.fit(X_train, y_train)
y_predict_train = linear_regressor.predict(X_train)


# In[ ]:


y_pre=svr_poly.fit(X,y).predict(X)


# In[33]:
```

```
X=np.array(al[:45])
X=X.reshape(-1, 1)
y=data2['10']


### 线性方法

# In[7]:


import numpy as np


# In[22]:


al


# In[110]:


data2


# In[121]:


X_train=np.array(al[:45])
X_train=X_train.reshape(-1, 1)
X_test=np.array([5519,5639,5759])
X_test=X_test.reshape(-1, 1)
d2_pre=pd.DataFrame()
for i in range(1,101):
    i=str(i)
```

```
    y_train=list(data2[i].values)
    linear_regressor = linear_model.LinearRegression()
    linear_regressor.fit(X_train, y_train)
    d2_pre[i] = linear_regressor.predict(X_test)


# In[122]:


d2_pre


# In[60]:


X_train=np.array(d.index)
X_train=X_train.reshape(-1, 1)
X_test=np.array([5579,5639,5699,5759])
X_test=X_test.reshape(-1, 1)
d_pre=pd.DataFrame()
for i in range(1,101):
    i=str(i)
    y_train=list(d[i].values)
    linear_regressor = linear_model.LinearRegression()
    linear_regressor.fit(X_train, y_train)
    d_pre[i] = linear_regressor.predict(X_test)
```

问题四模型

```
## 问题四
# In[64]:
al=[]
for i in range(1,49):
    al.append(120*i-1)
al

# In[65]:
```

```
#d=data2.drop(columns=['Time','total'])
d

d_30=pd.DataFrame()
for i in al:
    if i < 5519:
        #j=i-60
        j=(i+1)/120-1
        d_30[i]=d.T[j]#(d.T[i]+d.T[j])/2
    else:
        j=(i+1)/120-46
        d_30[i]=d2_pre.T[j]#(d_pre.T[i]+d_pre.T[j])/2
d_30

# In[73]:

pd.set_option('display.max_rows', 10)
d_30.T

# In[76]:

pd.set_option('display.max_columns',10)
d_30t=d_30.T
d_30t['total']=d_30.describe().T['mean']*100
d_30t

# In[85]:

d_30t['safe_score']=finals
d_30t

# In[78]:

asrs#=d_30.describe().T['mean']*100

# In[82]:
```

```
mins=min(asrs.values)
maxs=max(asrs.values)
finals=100-(asrs.values-mins)/(maxs-mins)*100

# In[86]:

plt.scatter(al, finals)
plt.show()
```