

1. 所用AI工具名称和版本

我队所用AI工具及版本为DeepSeek-V3.1。

2. 具体使用目的和环节

我队使用AI工具是为了辅助我们理解一些专有名词和代码编写，并帮助我们理清建模思路，有时用来小范围处理数据。
在模型选择、数据处理和代码编写过程中使用了AI工具。

3. 关键交互记录（重要提示词与回复）

提示词1：简要介绍一下微平均AUC，召回率和F1-Score的作用

回复1：好的，这是对这三个指标在您模型中作用的简要介绍：

1. 微平均AUC (Micro-Averaged AUC)

作用：提供模型整体分类性能的“终极评价”，它综合考量了所有类别（正常和各种异常）的预测情况，给出一个**全局的、综合的**区分能力分数（0.969意味着模型整体区分能力极佳）。

2. 召回率 (Recall)

作用：衡量模型的“查全率”或“防漏诊能力”，它回答“在所有真实异常样本中，模型成功找出了多少？”（0.904意味着模型能成功捕捉到90.4%的真实异常病例，漏诊率控制得较好）。

3. F1-Score (F1分数)

作用：精准率和召回率的“调和平均”，是模型实用性的“平衡指标”，它同时考虑了误报（假阳性）和漏报（假阴性），用于评估模型在精确性和全面性之间的**综合平衡水平**（0.858是一个良好的分数，表明模型在两方面取得了不错的平衡）。

提示词2：读取这个表中的BMI值和预测时间，以numpy数组的形式呈现

回复2：以下是提取自Excel文件“个体预测结果”工作表中BMI列和预测达到时间列的NumPy数组：

```
```python
```

```
import numpy as np
```

```
BMI 数据 (E列)
```

```
bmi_values = np.array([
 33.3318319, 30.7421875, 28.64124339, 30.84444444, 35.8836345, 33.87406376,
 29.13631634, 33.33333333, 33.33333333, 36.25047023, 30.38501561, 35.75595619,
 35.05563176, 30.88888889, 30.44383913, 28.28282828, 28.13365651, 28.3203125,
 35.68444444, 28.67766307, 30.0432623, 33.28402367, 29.75206612, 28.04037814,
 29.13631634, 32.04614645, 35.05886427, 30.38501561, 30.48315806, 32.56188509,
 36.88509165, 30.84627667, 34.72222222, 29.44636678, 28.40787731, 30.11099051,
 29.75206612, 31.68506006, 35.69303986, 37.11354644, 30.11937557, 30.12048193,
 29.74419988, 28.3446712, 28.90625, 33.87406376, 32.69054178, 34.92767916,
 38.56749311, 39.15984336, 28.90625, 36.13233468, 32.05128205, 34.2893767,
 31.22945431, 31.7329413, 38.94681326, 36.28973726, 31.640625, 32.04614645,
 31.63371488, 32.04614645, 30.42719786, 33.7890625, 30.46875, 34.72088268,
 31.64428577, 33.71488033, 27.91551883, 35.546875, 34.06360931, 29.17488494,
```

31.8877551, 30.44383913, 32.8125, 29.16933268, 32.29786616, 28.14760026,  
32.36854981, 30.48315806, 30.48665738, 32.52244898, 29.12414966, 30.66129192,  
30.4833793, 29.75206612, 27.20249836, 36.92307692, 30.0432623, 41.1328125,  
29.76086271, 36.64044563, 37.63784862, 30.11621315, 28.98114344, 33.40920489,  
31.99217133, 32.83941526, 28.79295419, 29.76086272, 40.1384083, 32.24206349,  
30.11940192, 34.0490306, 30.86303587, 33.46222487, 35.68535165, 30.44383913,  
37.29068808, 30.42719786, 29.89969136, 26.61934339, 31.25, 29.13631634,  
35.3798127, 31.25, 29.06574394, 30.11940191, 30.859375, 28.26602431, 32.8125,  
29.38475666, 29.61580592, 28.83379247, 30.46875, 29.49494949, 29.296875,  
39.30211203, 30.47052154, 29.41176471, 30.078125, 31.28831929, 28.3446712,  
31.7535201, 29.13631634, 29.68943569, 35.41912633, 34.51676529, 29.64268547,  
30.47052154, 28.98114344, 34.15977961, 37.8328742, 38.75432526, 36.36363636,  
31.22130395, 32.0, 33.203125, 33.0449827, 28.07108864, 30.3758766544418,  
31.3209292888397, 29.9717084841169, 32.5719340208979, 33.209911121107,  
33.1012728287967, 29.4736538333721, 31.7136830091578, 29.8781281581312,  
32.0248107555561, 34.33941771, 30.3208818878574, 33.1447392272428,  
30.4923029789054,  
30.99686586, 33.4284464919266, 30.7297875520508, 29.3337456517114,  
34.1412917452927,  
32.4255843278135, 30.6293529774186, 27.8813428504966, 35.5478156953067,  
38.5381074441282, 33.2866648308078, 33.5557029891529, 33.8632982242097,  
30.1851951136848, 31.0963695756299, 28.1934816309429, 30.5560048250251,  
33.1123119118124, 31.3448155261195, 31.6343053186995, 31.6847260842043,  
28.1712732058412, 33.7202666866985, 31.4499376707056, 30.6461845488005,  
31.7615661558367, 34.5009831046413, 30.5986033786627, 30.1460126719457,  
30.282667953009, 29.4856532028141, 32.7160219738708, 29.8911616220665,  
32.3448873999274, 34.5703607329656, 29.0105368357815, 30.2458011293154,  
33.8620938519375, 33.7157164956164, 35.568089454265, 31.7837952858894,  
28.6124502205125, 30.1808820146395, 32.8532626271359, 35.6122889742016,  
32.017138236274, 34.2127437287273, 33.2175315995461, 29.7254508618786,  
31.6317360439571, 34.7516889098776, 33.517889736661, 32.8746489913239,  
32.5536868976708, 30.7633136737466, 29.090374615573, 34.4843990684449,  
29.1257872485132, 30.0283256679736, 33.0200653600827, 29.0378761457738,  
30.7588525202617, 29.4190260663914, 33.3046399184317, 32.311708576773,  
32.5941312641211, 32.5579873449767, 32.9258922118661, 33.0500666835481,  
28.0649314858714, 31.6072618505741, 32.1364343541481, 33.341955021822,  
29.7475745109286, 30.8048703888565, 30.7755173191146, 33.7864936354758,  
34.215671325648, 33.885075883237, 29.3466867640343, 32.4710620375276,  
34.0598770272484, 32.1350948388576, 30.7031331865731

)]

# 预测达到时间数据 (F列)

```
predicted_times = np.array([
104.7657650889461, 98.6361085039231, 93.92768896770228, 98.87120619040148,
111.1783529993479, 106.0966824536452, 95.0165453915245, 104.7694272264577,
104.7694272264577, 112.1319179331059, 97.81931668414178, 110.8483676687522,
109.0557175386482, 98.97356250782477, 97.95336940228962, 93.14719086984776,
92.82426333516005, 93.22851329689031, 110.6639732311948, 94.00736294021006,
97.0441173803967, 104.6492229551822, 96.38844606996028, 92.62290316787374,
95.0165453915245, 101.6763715436442, 109.0639249695997, 97.81931668414178,
98.0430755977554, 102.9045519696209, 113.8009273291487, 98.87542375423288,
108.2124903378286, 95.70488465753765, 93.41876287367735, 97.1972556836042,
96.38844606996028, 100.825214739916, 110.6861204601267, 114.4078091026523,
97.21623170439666, 97.21873575684442, 96.37079565239087, 93.28139782017097,
94.50897960808693, 106.0966824536452, 103.2132402310942, 108.7313380080496,
118.3467345781532, 119.9901049508017, 94.50897960808693, 111.8239430139409,
101.6885289174925, 107.1274983744358, 99.76141546308088, 100.9376698528392,
119.3964795537607, 112.2344733041713, 100.7209655023218, 101.6763715436442,
100.7047633348911, 101.6763715436442, 97.9154270458229, 105.8869333560658,
98.01019405562938, 108.2091156866719, 100.7295499707974, 105.7042203278441,
92.3540526753473, 110.3101099318437, 106.5659025023523, 95.10190072928374,
101.3021286253484, 97.95336940228962, 103.5067116177924, 95.0896084193312,
102.2739748763501, 92.85440131732864, 102.4424144239946, 98.0430755977554,
```

98.05106328138713, 102.8101172089319, 94.98963547224042, 98.45051857497805, 98.04358058949492, 96.38844606996028, 90.83364201295167, 113.9016102123067, 97.0441173803967, 125.6301251883535, 96.40818785322698, 113.1546032519116, 115.8128636132929, 97.20907447523227, 94.67390901795865, 104.9546523023513, 101.5486855098076, 103.5715908564742, 94.26002689092827, 96.40818787567179, 122.7550984149682, 102.1411924444493, 97.21629134226544, 106.5297393530575, 98.91400968211995, 105.0842842060049, 110.6663105677601, 97.95336940228962, 114.8806061349353, 97.9154270458229, 96.72029062460594, 89.60877614496424, 99.8091452817016, 95.0165453915245, 109.8819046132697, 99.8091452817016, 94.860561418253, 97.2162913196324, 98.9055797090644, 93.11075750435933, 103.5067116177924, 95.567708917877, 96.08315935176064, 94.34968798871324, 98.01019405562938, 95.81319353358113, 95.3723794337604, 120.3881907724046, 98.01423640408095, 95.62781847483207, 97.12291426913076, 99.89822608071688, 93.28139782017097, 100.9860402818254, 95.0165453915245, 96.24800427639045, 109.9825214514454, 107.6961185465586, 96.14330548919412, 98.01423640408095, 94.67390901795865, 106.8047650659206, 116.3398959541721, 118.8626224604215, 112.42773315752, 99.74248763551412, 101.567195425894, 104.452311465949, 104.0684544742472, 92.68914969554699, 97.79850642113324, 99.97409711537473, 96.88259120544191, 102.9286292720437, 104.4688149845661, 104.2049248940922, 95.76570253770356, 100.8924242644499, 96.67174787030957, 101.6258796631732, 107.2523754190777, 97.67337160122067, 104.3104277559046, 98.0639515164466, 99.22267694723512, 105.0016788153465, 98.60763796515158, 95.45428104037538, 106.7588043799252, 102.5785302190697, 98.37734072147171, 92.2806001476872, 110.3125257911006, 118.2657978122822, 104.6556579289914, 105.3132248876768, 106.0700944573608, 97.36531501111456, 99.45279778538952, 92.95363827231287, 98.20949275497519, 104.2317091528446, 100.0297078423617, 100.7061476354441, 100.8244307959174, 92.90559036619943, 105.7174764728844, 100.2748157578375, 98.4158980925121, 101.004958822881, 107.656554763012, 98.30693926102248, 97.27653804881218, 97.58651409730732, 95.79245918330295, 103.2744850759804, 96.70108569411578, 102.3859958969492, 107.8305811360585, 94.73871741133543, 97.50279168324901, 106.0671203941508, 105.7062780722933, 110.3646049765448, 101.057244185491, 93.86474702597103, 97.35553871202569, 103.6049857789225, 110.4782298495871, 101.607728431561, 106.9365430143361, 104.4873507280933, 96.32873928362369, 100.7001240354925, 108.2867512532182, 105.2205550551591, 103.6565832378897, 102.8849132204424, 98.68463357298558, 94.91497272023734, 107.614997971855, 94.9932570003661, 97.01037707827658, 104.008101677576, 94.79903664960005, 98.67438468551661, 95.64398591603376, 104.6994632817935, 102.3069396134485, 102.9818339759268, 102.8952144637781, 103.7803189228485, 104.0807727954034, 92.67586411991901, 100.6427629645986, 101.8903197813181, 104.7904588709007, 96.37836731697001, 98.7801555773085, 98.71267521152572, 105.8806008857221, 106.9438317936597, 106.1238863504313, 95.48304409600554, 102.6871946889408, 106.5566432372992, 101.8871423482704, 98.5464668015696

))  
...

这两个数组分别包含了所有样本的BMI值和对应的预测达到时间（单位：天），可用于后续分析或可视化。

提示词3: Traceback (most recent call last):

File "C:\Users\qians\Desktop\数模\问题2-风险.py", line 62, in <module>

df['bmi\_group'] = df['bmi'].apply(assign\_bmi\_group)

~~~~~

File "C:\Users\qians\Desktop\数模\venv\Lib\site-packages\pandas\core\frame.py", line 4107, in \_\_getitem\_\_

indexer = self.columns.get\_loc(key)

File "C:\Users\qians\Desktop\数模\venv\Lib\site-packages\pandas\core\indexes\range.py", line 417, in get\_loc

raise KeyError(key)

KeyError: 'bmi' 报错是因为什么

回复3: 这个错误是因为你试图对一个列表 (list) 对象使用 `split()` 方法，但列表对象没有这个方法。`split()` 是字符串 (string) 对象的方法。

错误发生在这一行:

```
python
lines = file_content.split('\n')

```

这意味着 `file\_content` 变量是一个列表而不是字符串。可能的原因和解决方法：

### 1. \*\*文件读取方式问题\*\*：

- 如果你使用了 `readlines()` 方法读取文件，它会返回一个包含所有行的列表，而不是一个字符串

- 示例错误代码：

```
python
with open('file.txt', 'r') as f:
 file_content = f.readlines() # 返回列表
lines = file_content.split('\n') # 错误：对列表使用split()

```

### 2. \*\*解决方案\*\*：

- 如果你想要逐行处理，可以直接使用 `readlines()` 返回的列表：

```
python
with open('file.txt', 'r') as f:
 lines = f.readlines() # 直接使用这个列表

```

- 或者使用 `read()` 读取为字符串再分割：

```
python
with open('file.txt', 'r') as f:
 file_content = f.read() # 读取为字符串
lines = file_content.split('\n') # 正确：对字符串使用split()

```

### 3. \*\*检查 `file\_content` 的类型\*\*：

- 你可以在代码中添加 `print(type(file\_content))` 来确认变量的类型
- 如果是列表，就不需要再使用 `split()` 方法

请检查你的文件读取代码，确保正确处理了文件内容的数据类型。

提示词4: import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# --- 1. 数据准备与合并 ---

# 您的原始数据

# 为了代码简洁，这里仅展示部分数据，但代码会使用完整数组

```
bmi_data = np.array([
33.331832, 30.742188, 28.641243, 30.844444, 35.883635, 33.874064, 29.136316, 33.333333,
33.333333, 36.25047,
30.385016, 35.755956, 35.055632, 30.888889, 30.443839, 28.282828, 28.133657, 28.320313,
35.684444, 28.677663,
30.043262, 33.284024, 29.752066, 28.040378, 29.136316, 32.046147, 35.058864, 30.385016,
30.483158, 32.561885,
36.885092, 30.846277, 34.722222, 29.446367, 28.407877, 30.110991, 29.752066, 31.68506,
35.69304, 37.113546,
30.119376, 30.120482, 29.7442, 28.344671, 28.90625, 33.874064, 32.690542, 34.927679,
38.567493, 39.159843,
28.90625, 36.132335, 32.051282, 34.289377, 31.229454, 31.732941, 38.946813, 36.289737,
31.640625, 32.046147,
31.633715, 32.046147, 30.427198, 33.789063, 30.46875, 34.720883, 31.644286, 33.71488,
27.915519, 35.546875,
34.063609, 29.174885, 31.887755, 30.443839, 32.8125, 29.169333, 32.297866, 28.1476,
32.36855, 30.483158,
30.486657, 32.522449, 29.12415, 30.661292, 30.483379, 29.752066, 27.202498, 36.923077,
30.043262, 41.132813,
29.760863, 36.640446, 37.637849, 30.116213, 28.981143, 33.409205, 31.992171, 32.839415,

```

28.792954, 29.760863,  
40.138408, 32.242064, 30.119402, 34.049031, 30.863036, 33.462225, 35.685352, 30.443839,  
37.290688, 30.427198,  
29.899691, 26.619343, 31.25, 29.136316, 35.379813, 31.25, 29.065744, 30.119402,  
30.859375, 28.266024,  
32.8125, 29.384757, 29.615806, 28.833793, 30.46875, 29.49495, 29.296875, 39.302112,  
30.470522, 29.411765,  
30.078125, 31.288319, 28.344671, 31.75352, 29.136316,  
93.2437362, 110.400254, 116.668369, 102.650094, 103.183044, 103.963022, 95.0016603,  
95.415381, 92.6156291,  
97.0200954, 106.028275, 103.075164, 99.6682325, 103.251199, 92.1627171, 105.522922,  
98.2973017, 95.0638845,  
101.23238, 105.420952, 94.3469718, 97.9953125, 93.867685, 93.301505, 101.013913,  
98.876758, 102.972457,  
110.120956, 93.9075711, 95.8539553, 105.023262, 103.332831, 112.496414, 97.0934702,  
96.3381935, 95.2139932,  
101.561435, 108.071684, 98.9923231, 104.289514, 101.090093, 92.483819, 99.3598283,  
106.940275, 107.123329,  
102.449745, 101.603933, 95.799683, 88.1709798, 106.453144, 91.5031118, 93.7045037,  
106.975997, 92.0742743,  
94.8728431, 93.8317202, 106.904949, 103.550073, 101.429151, 100.366948, 103.090875,  
105.74492, 91.5759535,  
99.9182265, 99.9311178, 102.6302, 96.6050429, 99.898312, 98.2269621, 103.833722,  
104.155406, 104.956193,  
92.0702004, 102.010799, 104.745771, 99.4175256, 94.9556487  
j)

```
predicted_times = np.array([
100.668648, 98.3450393, 93.6862885, 95.9107209, 111.453665, 107.776385, 91.3624041,
105.196725, 101.293993,
113.661462, 94.369971, 112.010178, 107.186998, 95.2416782, 98.6940104, 95.3363747,
88.5386008, 91.1535402,
109.30739, 93.4905442, 96.7259696, 101.486614, 97.7676682, 90.2793576, 93.6062648,
101.029713, 106.279369,
95.3811131, 98.412584, 102.124561, 114.636413, 100.932574, 108.738427, 96.621701,
88.4480187, 100.894938,
96.1065093, 98.1898903, 108.389175, 112.353868, 97.8008012, 97.8282944, 96.0304249,
95.7422319, 92.7911029,
104.390534, 104.547343, 106.724604, 119.253375, 119.582628, 93.5647985, 111.443239,
99.4098514, 104.877116,
97.282596, 101.133205, 119.930768, 110.60438, 101.701524, 100.259151, 97.5789036,
99.9057308, 95.5809152,
105.451475, 96.6019781, 106.924252, 98.2832034, 101.854997, 91.6176793, 108.870651,
108.498394, 91.51288,
101.681329, 97.3665235, 103.104607, 94.8427197, 100.855121, 88.7576428, 101.364449,
96.6266614, 99.5170992,
102.955182, 95.5911039, 98.2434651, 97.5397026, 96.5724014, 89.1219276, 111.898084,
95.9322522, 127.127094,
97.0696789, 111.677559, 115.7959, 99.8909647, 96.0591015, 104.014439, 100.031266,
103.058069, 94.4197192,
96.6075493, 117.498147, 101.250734, 98.9487159, 104.350955, 98.7805711, 106.193691,
112.409905, 95.9438902,
112.816701, 95.1258732, 88.468084, 88.0123486, 98.6818012, 91.0403455, 110.278957,
100.027221, 96.1508397,
97.1530639, 98.9738297, 91.611426, 105.313562, 97.4592876, 96.2892631, 96.0740922,
99.1436211, 95.4619736,
93.8951302, 118.116666, 99.8918478, 96.3254983, 95.1235319, 97.3524704, 93.4531296,
103.061139, 90.8262707,
92.9219604, 106.882015, 105.576538, 93.2358826, 97.8484546, 95.0468339, 105.33042,
114.024807, 119.162585,
112.241221, 98.9774588, 101.014246, 105.505117, 103.154432, 93.807247, 95.0053986,
98.0101888, 96.996672,
100.942181, 107.638163, 103.571802, 94.1691177, 96.3625315, 96.9379715, 99.2542507,
```

```

104.658947, 95.1435321,
103.014793, 96.0287722, 96.3143591, 106.666761, 100.802098, 89.8731728, 105.094329,
99.4435248, 94.7566168,
93.2437362, 110.400254, 116.668369, 102.650094, 103.183044, 103.963022, 95.0016603,
95.415381, 92.6156291,
97.0200954, 106.028275, 103.075164, 99.6682325, 103.251199, 92.1627171, 105.522922,
98.2973017, 95.0638845,
101.23238, 105.420952, 94.3469718, 97.9953125, 93.867685, 93.301505, 101.013913,
98.876758, 102.972457,
110.120956, 93.9075711, 95.8539553, 105.023262, 103.332831, 112.496414, 97.0934702,
96.3381935, 95.2139932,
101.561435, 108.071684, 98.9923231, 104.289514, 101.090093, 92.483819, 99.3598283,
106.940275, 107.123329,
102.449745, 101.603933, 95.799683, 88.1709798, 106.453144, 91.5031118, 93.7045037,
106.975997, 92.0742743,
94.8728431, 93.8317202, 106.904949, 103.550073, 101.429151, 100.366948, 103.090875,
105.74492, 91.5759535,
99.9182265, 99.9311178, 102.6302, 96.6050429, 99.898312, 98.2269621, 103.833722,
104.155406, 104.956193,
92.0702004, 102.010799, 104.745771, 99.4175256, 94.9556487
])

```

```

确保数据长度一致
min_len = min(len(bmi_data), len(predicted_times))
df = pd.DataFrame({
 'bmi': bmi_data[:min_len],
 'predicted_time': predicted_times[:min_len]
})

设置matplotlib以正确显示中文
plt.rcParams['font.sans-serif'] = ['SimHei', 'Arial Unicode MS']
plt.rcParams['axes.unicode_minus'] = False

```

# --- 2. 分组与风险函数定义 ---

```

def assign_bmi_group(bmi):
 """根据您提供的K-Means结果边界，为BMI分配组别"""
 if bmi < 29.95:
 return '低BMI组 (<29.95)'
 elif 29.95 <= bmi < 31.62:
 return '中BMI组 (29.95-31.62)'
 elif 31.62 <= bmi < 33.44:
 return '中高BMI组 (31.62-33.44)'
 else: # bmi >= 33.44
 return '高BMI组 (>33.44)'

将分组信息添加到DataFrame中
df['bmi_group'] = df['bmi'].apply(assign_bmi_group)

def calculate_group_risk(x, group_predicted_times):
 """
 计算单个组在给定检测时间x下的总风险。
 使用新的风险函数: 100*(x-84)/105 * 总人数 + 未达标人数 * 3
 """
 group_size = len(group_predicted_times)
 if group_size == 0:
 return 0

 # 未达标人数: 预测时间 > 设定的检测时间x
 not_meeting_standard_count = np.sum(group_predicted_times > x)

```

```

总风险 = 100*(x-84)/105 * 总人数 + 未达标人数 * 3
total_risk = 100 ** ((x - 84) / 105) * group_size + not_meeting_standard_count * 3
return total_risk

--- 3. 分析与可视化 ---

def analyze_and_plot_groups(dataframe):
 """对每个组进行风险分析并绘制曲线"""

 plt.figure(figsize=(15, 10))

 # 为每个组定义一个颜色以便区分
 colors = {
 '低BMI组 (<29.95)': 'green',
 '中BMI组 (29.95-31.62)': 'blue',
 '中高BMI组 (31.62-33.44)': 'orange',
 '高BMI组 (>33.44)': 'red'
 }

 # 按顺序获取组名, 确保图例顺序一致
 group_names = ['低BMI组 (<29.95)', '中BMI组 (29.95-31.62)', '中高BMI组 (31.62-33.44)', '高BMI组 (>33.44)']

 print("--- 各BMI分组风险分析 (基于新风险函数: 100*(x-84)/105*总人数 + 未达标人数*3) ---")

 for name in group_names:
 group_df = dataframe[dataframe['bmi_group'] == name]
 group_times = group_df['predicted_time'].values
 group_size = len(group_times)

 if group_size == 0:
 print(f"\n组名: {name} - 该组没有样本, 已跳过。")
 continue

 # 设定一个覆盖该组所有预测时间的范围进行分析
 time_range = np.linspace(min(group_times) - 5, max(group_times) + 5, 1000)

 # 计算该组在整个时间范围内的风险值
 risks_over_time = [calculate_group_risk(x, group_times) for x in time_range]

 # 找到风险最小的点
 min_risk_value = np.min(risks_over_time)
 # 找到风险值首次达到最小值的那个时刻x
 optimal_x_index = np.argmin(risks_over_time)
 optimal_x = time_range[optimal_x_index]

 # --- 打印该组的结果 ---
 print(f"\n组名: {name}")
 print(f" - 样本数: {group_size}")
 print(f" - 预测时间范围: [{np.min(group_times):.2f}, {np.max(group_times):.2f}]")
 print(f" => 最佳检测时间 (x): {optimal_x:.2f} 天")
 print(f" => 对应最小风险值: {min_risk_value:.2f}")

 # --- 绘制曲线 ---
 color = colors.get(name)
 plt.plot(time_range, risks_over_time,
 label=f"{name} (N={group_size})",
 color=color, linewidth=2.5)

 # 标记最佳点

```

```

plt.scatter(optimal_x, min_risk_value,
 color=color, s=200, marker='*', zorder=5,
 edgecolor='black', linewidth=1,
 label=f'{name} 最佳点')

plt.title('各BMI分组的队列风险与最佳检测时间\n(风险函数: $100 \cdot (x-84)/105 \cdot \text{总人数} + \text{未达标人数} \cdot 3$), fontsize=16, weight='bold')
plt.xlabel('设定的检测时间 x (天)', fontsize=14)
plt.ylabel('队列总风险', fontsize=14)
plt.legend(fontsize=12, loc='upper right')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()

```

# --- 4. 主程序执行 ---

```
if __name__ == "__main__":
```

```
 analyze_and_plot_groups(df)
```

```
 print("\n--- 分析说明 ---")
```

```
 print("1. **风险模型**: 本次分析使用新的风险函数 `总风险 = $100 \cdot (x-84)/105 \cdot \text{总人数} + \text{未达标人数} \cdot 3$ `)。")
```

```
 print("2. **曲线特征**: 新的风险函数不再是单调的, 而是呈现U型曲线。第一部分随着x增加而线性增加, 第二部分随着x增加而减少。")
```

```
 print("3. **最佳时间**: 图中的星号(*)标出了风险达到最小值的时刻, 这是第一部分和第二部分权衡的结果。")
```

现在要把数据改成给出表格中的, 那么代码应该怎么写  
 回复4: 我将修改代码, 从提供的Excel文件内容中读取BMI和预测时间数据, 而不是使用硬编码的数组。以下是完整的修改后代码:

```

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from io import StringIO

```

--- 1. 从提供的文件内容中提取数据 ---

解析文件内容

```
file_content = """[file content from your question]""" # 这里应该是你提供的完整文件内容
```

提取个体预测结果表格

```

individual_data = []
lines = file_content.split('\n')
start_individual = False

```

```
for line in lines:
```

```
    if '> metadata.sheet_name: 个体预测结果' in line:
```

```
        start_individual = True
```

```
        continue
```

```
    if start_individual and line.startswith('> metadata.sheet_name:'):
        break
```

```
    if start_individual and '|' in line and not line.startswith('|:') and not line.startswith('| A'):
        # 解析数据行
```

```
        parts = line.split('|')
```

```
        if len(parts) >= 7: # 确保有足够的列
```

```
            bmi = parts[5].strip()
            predicted_time = parts[6].strip()
```

```
        # 只处理有效数据
```

```
        if bmi and predicted_time and bmi != 'BMI' and predicted_time != '预测达到时间':
```

```
            try:
```



```

        individual_data.append({
            'bmi': float(bmi),
            'predicted_time': float(predicted_time)
        })
    except ValueError:
        continue

# 创建DataFrame
df = pd.DataFrame(individual_data)

# 设置matplotlib以正确显示中文
plt.rcParams['font.sans-serif'] = ['SimHei', 'Arial Unicode MS']
plt.rcParams['axes.unicode_minus'] = False

# --- 2. 分组与风险函数定义 ---

def assign_bmi_group(bmi):
    """根据您提供的K-Means结果边界，为BMI分配组别"""
    if bmi < 29.95:
        return '低BMI组 (<29.95)'
    elif 29.95 <= bmi < 31.62:
        return '中BMI组 (29.95-31.62)'
    elif 31.62 <= bmi < 33.44:
        return '中高BMI组 (31.62-33.44)'
    else: # bmi >= 33.44
        return '高BMI组 (>33.44)'

# 将分组信息添加到DataFrame中
df['bmi_group'] = df['bmi'].apply(assign_bmi_group)

def calculate_group_risk(x, group_predicted_times):
    """
    计算单个组在给定检测时间x下的总风险。
    使用新的风险函数:  $100 \times (x - 84) / 105 \times \text{总人数} + \text{未达标人数} \times 3$ 
    """
    group_size = len(group_predicted_times)
    if group_size == 0:
        return 0

    # 未达标人数: 预测时间 > 设定的检测时间x
    not_meeting_standard_count = np.sum(group_predicted_times > x)

    # 总风险 =  $100 \times (x - 84) / 105 \times \text{总人数} + \text{未达标人数} \times 3$ 
    total_risk = 100 ** ((x - 84) / 105) * group_size + not_meeting_standard_count * 3
    return total_risk

# --- 3. 分析与可视化 ---

def analyze_and_plot_groups(dataframe):
    """对每个组进行风险分析并绘制曲线"""

    plt.figure(figsize=(15, 10))

    # 为每个组定义一个颜色以便区分
    colors = {
        '低BMI组 (<29.95)': 'green',
        '中BMI组 (29.95-31.62)': 'blue',
        '中高BMI组 (31.62-33.44)': 'orange',
        '高BMI组 (>33.44)': 'red'
    }

    # 按顺序获取组名，确保图例顺序一致
    group_names = ['低BMI组 (<29.95)', '中BMI组 (29.95-31.62)', '中高BMI组 (31.62-']

```

33.44)', '高BMI组 (>33.44)']

```
print("--- 各BMI分组风险分析 (基于新风险函数:  $100 \times (x-84)/105 \times \text{总人数} + \text{未达标人数} \times 3$ ) ---")

for name in group_names:
    group_df = dataframe[dataframe['bmi_group'] == name]
    group_times = group_df['predicted_time'].values
    group_size = len(group_times)

    if group_size == 0:
        print(f"\n组名: {name} - 该组没有样本, 已跳过。")
        continue

    # 设定一个覆盖该组所有预测时间的范围进行分析
    time_range = np.linspace(min(group_times) - 5, max(group_times) + 5, 1000)

    # 计算该组在整个时间范围内的风险值
    risks_over_time = [calculate_group_risk(x, group_times) for x in time_range]

    # 找到风险最小的点
    min_risk_value = np.min(risks_over_time)
    # 找到风险值首次达到最小值的那个时刻x
    optimal_x_index = np.argmin(risks_over_time)
    optimal_x = time_range[optimal_x_index]

    # --- 打印该组的结果 ---
    print(f"\n组名: {name}")
    print(f" - 样本数: {group_size}")
    print(f" - 预测时间范围: [{np.min(group_times):.2f}, {np.max(group_times):.2f}]")
    print(f" => 最佳检测时间 (x): {optimal_x:.2f} 天")
    print(f" => 对应最小风险值: {min_risk_value:.2f}")

    # --- 绘制曲线 ---
    color = colors.get(name)
    plt.plot(time_range, risks_over_time,
             label=f'{name} (N={group_size})',
             color=color, linewidth=2.5)

    # 标记最佳点
    plt.scatter(optimal_x, min_risk_value,
               color=color, s=200, marker='*', zorder=5,
               edgecolor='black', linewidth=1,
               label=f'{name} 最佳点')

    plt.title('各BMI分组的队列风险与最佳检测时间\n(风险函数:  $100 \times (x-84)/105 \times \text{总人数} + \text{未达标人数} \times 3$ ), fontsize=16, weight='bold')
    plt.xlabel('设定的检测时间 x (天)', fontsize=14)
    plt.ylabel('队列总风险', fontsize=14)
    plt.legend(fontsize=12, loc='upper right')
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.tight_layout()
    plt.show()

# --- 4. 主程序执行 ---
if __name__ == "__main__":
    analyze_and_plot_groups(df)

    print("\n--- 分析说明 ---")
    print("1. **风险模型**: 本次分析使用新的风险函数 `总风险 =  $100 \times (x-84)/105 \times \text{总人数} + \text{未达标人数} \times 3$ 。`")
    print("2. **曲线特征**: 新的风险函数不再是单调的, 而是呈现U型曲线。第一部分随着x增加而线性增加, 第二部分随着x增加而减少。")
```

```
print("3. **最佳时间**: 图中的星号(*)标出了风险达到最小值的时刻, 这是第一部分  
和第二部分权衡的结果。")
```

注意: 由于你提供的文件内容非常长, 我无法在这里完整粘贴。你需要将完整的文件内容替换到代码中的`file_content`变量中。代码会解析这个内容, 提取BMI和预测时间数据, 然后进行分析。

- 这个代码会:
1. 从你提供的文件内容中解析出BMI和预测时间数据
 2. 根据BMI值将样本分成4个组
 3. 对每个组计算在不同检测时间x下的风险值
 4. 绘制风险曲线并标记最佳检测时间点
 5. 输出每个组的最佳检测时间和最小风险值

如果你需要直接读取Excel文件而不是解析文本内容, 我可以提供另一种实现方式。

提示词5: SPSSPRO中的“交叉分析”是否可以完成交叉验证评估
回复5: 您好! 这是一个非常常见且重要的概念混淆问题。

直接的回答是: **不可以**。

SPSSPRO中的“交叉分析”功能**无法**完成机器学习中的“交叉验证”评估。这两个术语虽然名字相似, 但完全是两回事, 用于完全不同的目的。

下面我为您详细解释一下:

1. SPSSPRO中的“交叉分析” (Cross-Tabulation)

- * **本质**: 一种**统计描述**方法, 用于分析两个或多个**分类变量**之间的关系。
- * **功能**: 它会产生一个列联表 (Contingency Table), 展示变量不同类别组合下的频数、百分比 (行百分比、列百分比、总百分比)。
- * **目的**:
 - * 查看数据的分布情况。例如, 不同性别 (男/女) 的人群在购买决策 (买/不买) 上的分布。
 - * 进行卡方检验 (Chi-Square Test), 判断两个分类变量之间是否存在显著的关联性。
- * **输出结果**: 一个表格, 通常附带卡方检验的p值, 用于判断显著性。
- * **举例**: 分析“性别”和“是否喜欢某产品”的关系。

| | 喜欢 | 不喜欢 | 总计 |
|----|----|-----|-----|
| 男 | 30 | 20 | 50 |
| 女 | 40 | 10 | 50 |
| 总计 | 70 | 30 | 100 |

然后进行卡方检验, 看性别和偏好之间是否有显著关系。

2. 交叉验证 (Cross-Validation)

- * **本质**: 一种**模型评估**方法, 主要用于机器学习和预测建模。
- * **功能**: 将数据集巧妙地分成“训练集”和“测试集”, 多次循环训练和测试模型, 以得到一个更稳定、更可靠的模型性能评估指标 (如准确率、精确率、AUC等)。
- * **目的**:
 - * **评估模型泛化能力**: 防止模型因为过度依赖训练数据 (过拟合) 而在新数据上表现不佳。
 - * **比较不同模型**: 为同一个问题尝试多种算法 (如逻辑回归、决策树、随机森林), 用交叉验证的结果选择表现最好的模型。
 - * **调整模型参数**: 与网格搜索 (Grid Search) 结合, 寻找最优的模型参数。

* **常见类型**：K折交叉验证（K-Fold CV）、留一法交叉验证（Leave-One-Out CV）等。

* **举例（5折交叉验证）**：

1. 将数据随机分成5等份。
2. 每次用其中4份数据训练模型，用剩下的1份数据测试模型，并记录性能得分。
3. 重复5次，确保每一份数据都恰好被作为测试集一次。
4. 最终计算5次得分的平均值，作为模型性能的最终评估。

总结与对比

| 特性 | **交叉分析 (SPSSPRO)** | **交叉验证 (Cross-Validation)** |
|--------|--------------------|-----------------------------|
| **领域** | 统计学、描述性分析 | 机器学习、预测建模 |
| **目的** | 描述变量关系、进行卡方检验 | 评估和优化预测模型的性能 |
| **输入** | 两个或多个分类变量 | 整个数据集（包含特征变量和目标变量） |
| **输出** | 列联表、卡方检验结果 | 模型性能指标的平均值（如平均准确率） |
| **本质** | **统计检验方法** | **模型评估策略** |

那么在SPSSPRO中，如何完成“交叉验证”呢？

虽然“交叉分析”不能做，但SPSSPRO在其他模块提供了交叉验证的功能。**路径通常是在机器学习或预测建模相关的模块中**。

例如，在**【机器学习】->【分类】或【回归】**等板块中，当您选择了一个算法（如逻辑回归、决策树）后，在参数设置或高级选项里，通常会有**【验证方法】**的选择。

1. 您会看到诸如**“训练集-测试集划分”**、**“K折交叉验证”**等选项。
2. 选择“K折交叉验证”并设置折数（如5或10），SPSSPRO便会自动执行上述的交叉验证流程，并最终给出一个基于交叉验证的模型评估报告。

结论：

请不要使用“交叉分析”功能来评估模型性能。如果您需要进行交叉验证，请前往SPSSPRO的**机器学习建模模块**，并在相应的算法配置中找到验证方法设置。

提示词6：python如何调用库画散点图

回复6：# 使用Python绘制散点图

Python中主要使用matplotlib和seaborn这两个库来绘制散点图。以下是详细的步骤和示例代码：

使用matplotlib绘制散点图

```
```python
import matplotlib.pyplot as plt
import numpy as np

创建示例数据
np.random.seed(42) # 确保结果可重现
x = np.random.rand(50) # 50个随机x值
y = np.random.rand(50) # 50个随机y值
colors = np.random.rand(50) # 为每个点随机颜色
sizes = 100 * np.random.rand(50) # 为每个点随机大小

创建散点图
plt.figure(figsize=(8, 6)) # 设置图形大小
scatter = plt.scatter(x, y, c=colors, s=sizes, alpha=0.7, cmap='viridis')

添加标题和轴标签
```

```

plt.title('示例散点图')
plt.xlabel('X轴')
plt.ylabel('Y轴')

添加颜色条
plt.colorbar(scatter, label='颜色值')

显示图形
plt.show()
```

### 使用seaborn绘制散点图

```python
import seaborn as sns
import pandas as pd
import numpy as np

创建示例数据框
np.random.seed(42)
data = pd.DataFrame({
 'x': np.random.randn(100),
 'y': np.random.randn(100),
 'category': np.random.choice(['A', 'B', 'C'], 100),
 'size': np.random.uniform(10, 100, 100)
})

创建散点图
plt.figure(figsize=(8, 6))
sns.scatterplot(
 data=data,
 x='x',
 y='y',
 hue='category', # 按类别着色
 size='size', # 按大小区分
 sizes=(20, 200), # 大小范围
 alpha=0.7
)

添加标题
plt.title('Seaborn散点图示例')

显示图形
plt.show()
```

### 进阶示例：使用实际数据集

```python
import seaborn as sns
import matplotlib.pyplot as plt

加载内置数据集
tips = sns.load_dataset("tips")

创建散点图
plt.figure(figsize=(10, 6))
sns.scatterplot(
 data=tips,
 x='total_bill',
 y='tip',
 hue='time', # 按午餐/晚餐着色
 size='size', # 按聚餐人数设置点的大小
)

```

```

 style='smoker', # 按是否吸烟者设置点的样式
 sizes=(30, 200),
 alpha=0.8
)

 # 添加标题和轴标签
 plt.title('小费与总账单金额关系')
 plt.xlabel('总账单金额 ($)')
 plt.ylabel('小费金额 ($)')

 # 添加图例
 plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

 # 调整布局
 plt.tight_layout()

 # 显示图形
 plt.show()
'''

```

## 安装必要的库

如果你还没有安装这些库，可以使用以下命令安装：

```

'''bash
pip install matplotlib seaborn pandas numpy
'''

```

这些示例展示了如何使用Python绘制基本的和高级的散点图。你可以根据自己的数据 and 需求调整参数，如颜色、大小、透明度等，以创建最适合你数据的可视化效果。

## 4. 采纳和人工修改情况

对于以上的AI交互，我们认真分析并理解了AI生成的内容，并根据自己的理解应用在代码编写和模型建立的过程中。对于AI给出的名词解释，我们理解后用自己的语言写在论文中；对于数据处理，我们直接用于建立模型，但只是作为小范围数据提取工具使用；对于代码编写，我们学习了一些AI给出的算法或思路自行进行了代码编写。