

一

二

三

四

五

六

七

八

九

十

十一 附录

附录 1: 支撑材料, 文件列表附录 2: 补充表格, 图片附录 3: 代码

附录 1: 支撑材料文件列表

- 问题一 • 混合模型 (R 语言代码)
- 问题二 • C(2)(权重计算).py
- C(2)(生存函数).py
 - C(2)(正态分布检验).py
 - C(2) 测试误差.py
 - C(2)(权重计算).py
 - C(2) 聚类.py
 - 不同组的累积达标曲线、效用函数曲线及最优时点标注图
 - 不同组的误差模拟 (时点分布、效用值分布及风险分布) 图
- 问题三 • 模拟退火算法
- 问题四 • 打折统计图
- 蔬菜品类销售量灰色关联分析
 - 退货统计图

附录 2: 补充表格, 图片

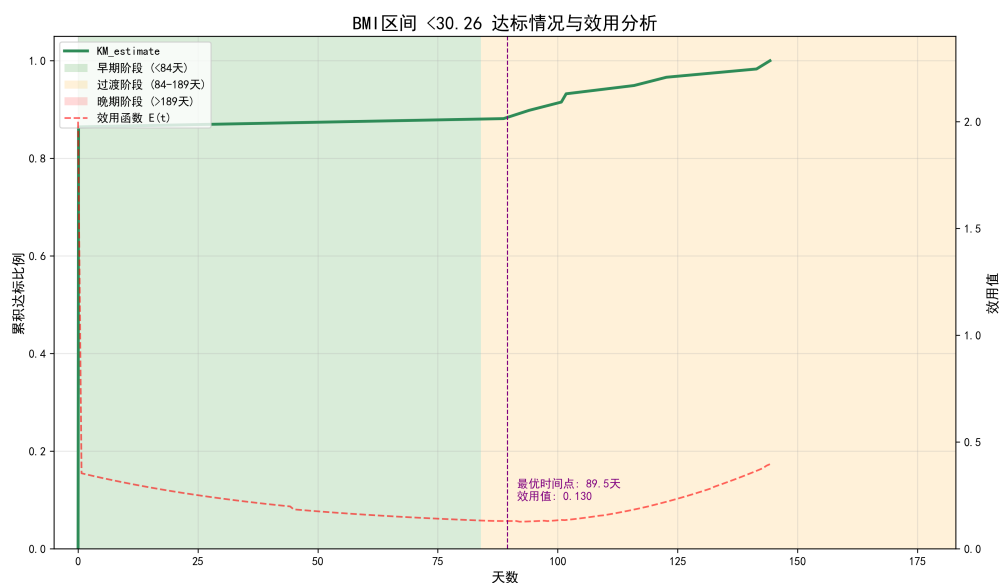


图 1: 累积达标曲线 (小于 30.26 组)

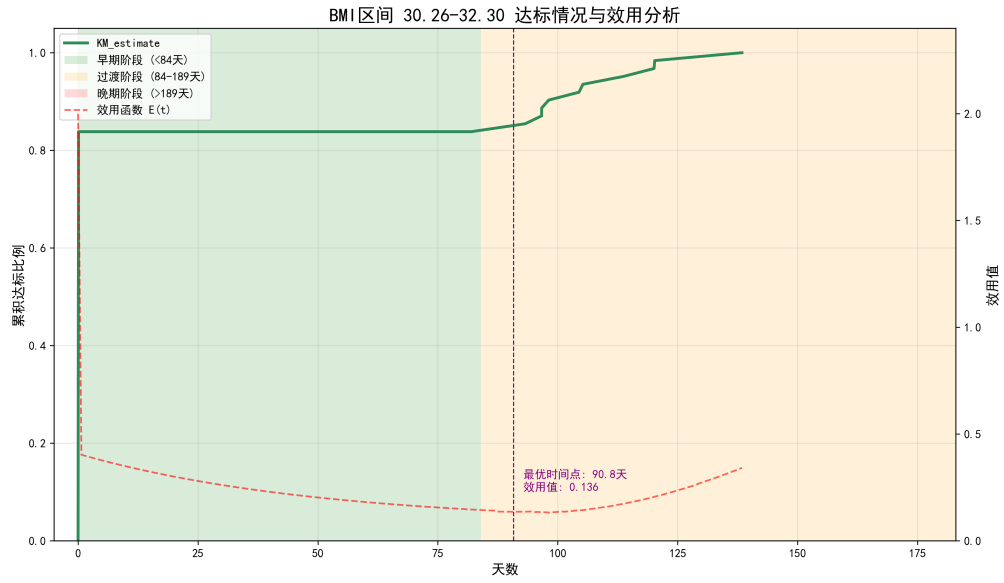


图 2: 累积达标曲线 (30.26-32.30 组)

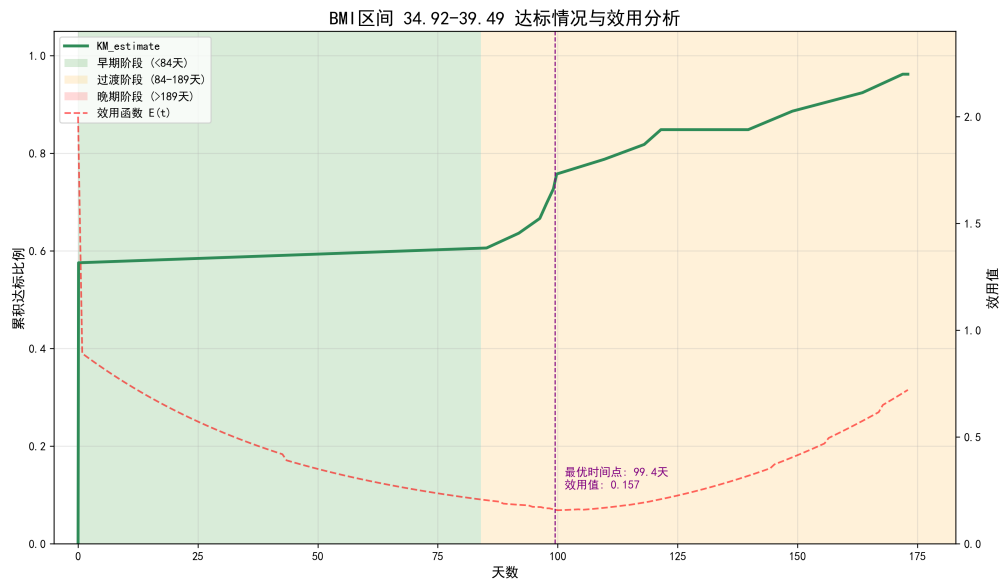


图 3: 累积达标曲线 (34.92-39.49 组)

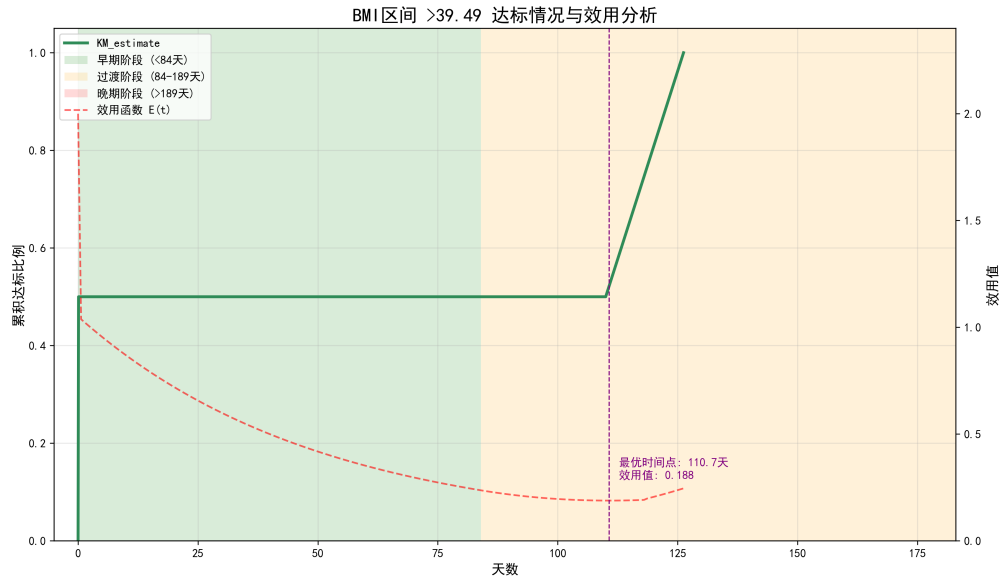


图 4: 累积达标曲线（大于 39.49 组）

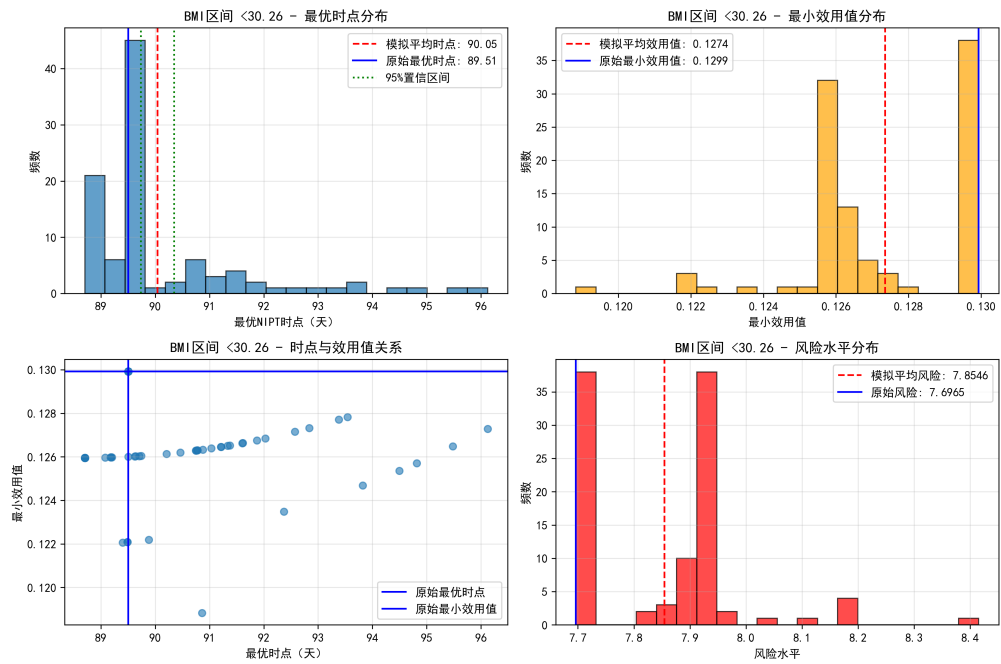


图 5: 误差模拟（小于 30.26 组）

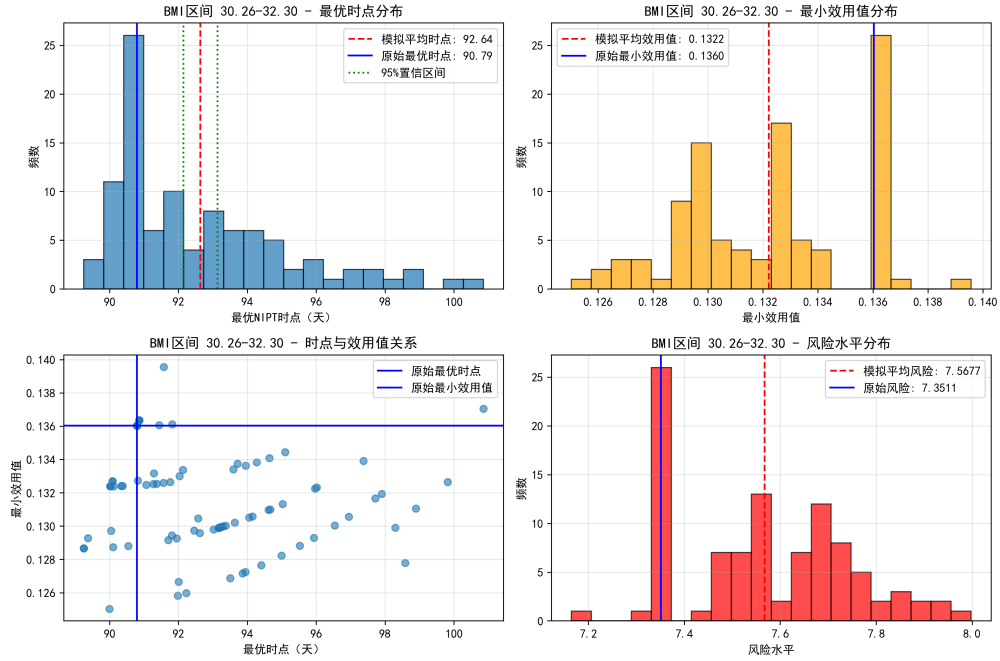


图 6: 误差模拟 (30.26-32.30 组)

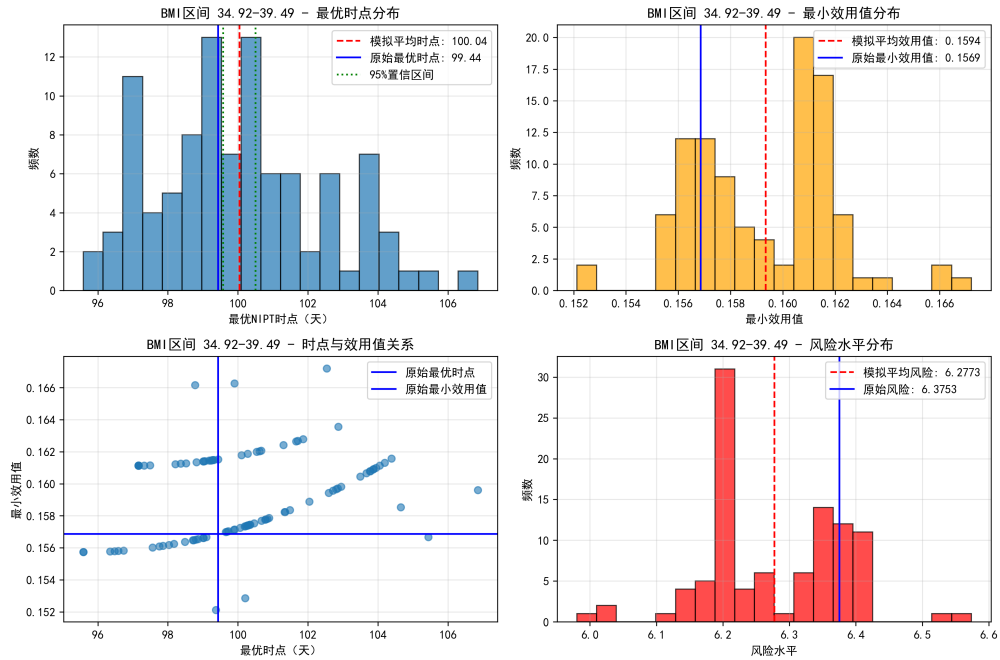


图 7: 误差模拟 (34.92-39.49 组)

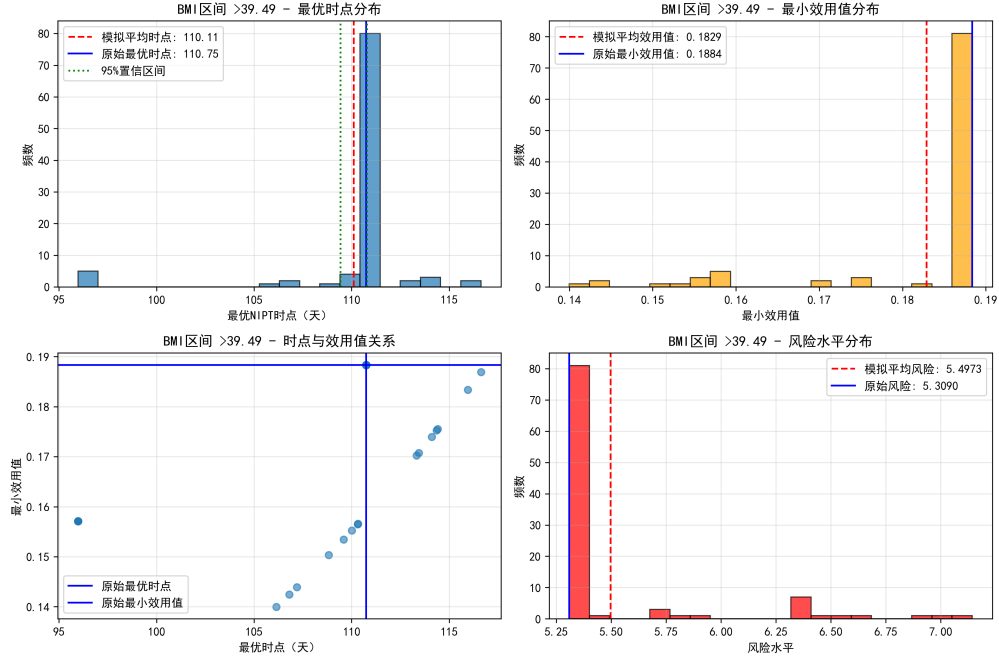


图 8: 误差模拟 (大于 39.49 组)

Listing 1: 混合模型.py

```

1  # 加载必要的包
2  library(readxl)
3  library(nlme)
4  library(ggplot2)
5  library(lme4)
6  library(MuMIn)
7  library(mgcv) # 用于GAMM模型
8  library(merTools)
9  library(sjPlot)
10 library(gridExtra)
11 library(itsadug) # 用于GAM模型可视化
12
13 # 读取数据
14 data_path <- "D:/math_modeling/25_C/mathmodeling_2024(1)/文件/副本C(1)_total.xlsx"
15 data <- read_excel(data_path)
16
17 # 查看数据结构
18 str(data)
19 head(data)
20
21 # 重命名列名以便使用
22 colnames(data) <- c("Subject", "BMI", "Y_concentration", "Gestational_days")
23
24 # 数据预处理
25 data$Subject <- as.factor(data$Subject)
26 data$BMI <- as.numeric(data$BMI)
27 data$Y_concentration <- as.numeric(data$Y_concentration)
28 data$Gestational_days <- as.numeric(data$Gestational_days)

```

```

29
30 # 探索性数据分析
31 exploratory_plot <- ggplot(data, aes(x = Gestational_days, y = Y_concentration, color = Subject)) +
32   geom_point() +
33   geom_smooth(method = "lm", se = FALSE) +
34   theme_minimal() +
35   labs(title = "Y染色体浓度随孕周变化", x = "孕周(天)", y = "Y染色体浓度")
36
37 print(exploratory_plot)
38 ggsave("exploratory_plot.png", width = 10, height = 6, dpi = 300)
39
40 # 1. 线性混合模型 (LMM)
41 cat("拟合线性混合模型...\n")
42 lmm_model <- lmer(Y_concentration ~ BMI + Gestational_days + (1|Subject), data = data)
43 cat("线性混合模型拟合完成\n")
44 print(summary(lmm_model))
45
46 # 2. 指数模型
47 cat("拟合指数模型...\n")
48 exp_model_formula <- function(bmi, days, a, b, c) {
49   a * exp(b * bmi + c * days)
50 }
51
52 tryCatch({
53   exp_nlme <- nlme(Y_concentration ~ exp_model_formula(BMI, Gestational_days, a, b, c),
54     data = data,
55     fixed = a + b + c ~ 1,
56     random = a ~ 1 | Subject,
57     start = list(a = 0.05, b = 0.001, c = 0.001),
58     control = nlmeControl(maxIter = 2000, pnlsTol = 1e-4, msTol = 1e-4))
59   cat("指数模型拟合成功\n")
60   print(summary(exp_nlme))
61 }, error = function(e) {
62   message("指数模型拟合失败: ", e$message)
63   exp_nlme <- NULL
64 })
65
66 # 3. Logistic生长模型
67 cat("拟合Logistic模型...\n")
68 logistic_model_formula <- function(bmi, days, a, b, c, d) {
69   a / (1 + exp(-(b * bmi + c * days - d)))
70 }
71
72 tryCatch({
73   logistic_nlme <- nlme(Y_concentration ~ logistic_model_formula(BMI, Gestational_days, a, b, c, d),
74     data = data,
75     fixed = a + b + c + d ~ 1,
76     random = a ~ 1 | Subject,
77     start = list(a = 0.1, b = 0.01, c = 0.01, d = 50),
78     control = nlmeControl(maxIter = 2000, pnlsTol = 1e-4, msTol = 1e-4))
79   cat("Logistic模型拟合成功\n")
80   print(summary(logistic_nlme))
81 }, error = function(e) {
82   message("Logistic模型拟合失败: ", e$message)
83   logistic_nlme <- NULL

```

```

84 })
85
86 # 4. Gompertz 生长模型
87 cat("拟合 Gompertz 模型...\n")
88 gompertz_model_formula <- function(bmi, days, a, b, c, d) {
89   a * exp(-exp(-(b * bmi + c * days - d)))
90 }
91
92 tryCatch({
93   gompertz_nlme <- nlme(Y_concentration ~ gompertz_model_formula(BMI, Gestational_days, a, b, c, d),
94     data = data,
95     fixed = a + b + c + d ~ 1,
96     random = a ~ 1 | Subject,
97     start = list(a = 0.1, b = 0.01, c = 0.01, d = 50),
98     control = nlmeControl(maxIter = 2000, pnlsTol = 1e-4, msTol = 1e-4))
99   cat("Gompertz 模型拟合成功\n")
100   print(summary(gompertz_nlme))
101 }, error = function(e) {
102   message("Gompertz 模型拟合失败: ", e$message)
103   gompertz_nlme <- NULL
104 })
105
106 # 5. 广义加性混合模型 (GAMM)
107 cat("拟合广义加性混合模型...\n")
108 tryCatch({
109   # 使用 bam 函数拟合 GAMM, 适用于大型数据集
110   gamm_model <- bam(Y_concentration ~ s(BMI) + s(Gestational_days) + s(Subject, bs = "re"),
111     data = data,
112     method = "REML")
113   cat("GAMM 模型拟合成功\n")
114   print(summary(gamm_model))
115
116   # 绘制平滑项图
117   png("gamm_smooth_terms.png", width = 10, height = 6, units = "in", res = 300)
118   par(mfrow = c(1, 2))
119   plot(gamm_model, select = 1, main = "BMI 平滑项")
120   plot(gamm_model, select = 2, main = "孕周平滑项")
121   dev.off()
122 }, error = function(e) {
123   message("GAMM 模型拟合失败: ", e$message)
124   gamm_model <- NULL
125 })
126
127 # 模型比较
128 cat("开始模型比较...\n")
129 model_comparison <- data.frame(
130   Model = "线性混合模型",
131   AIC = AIC(lmm_model),
132   BIC = BIC(lmm_model),
133   LogLik = as.numeric(logLik(lmm_model))
134 )
135
136 # 添加非线性模型 (如果成功拟合)
137 if(exists("exp_nlme") && !is.null(exp_nlme)) {
138   model_comparison <- rbind(model_comparison,

```



```

139         data.frame(Model = "指数模型",
140                     AIC = AIC(exp_nlme),
141                     BIC = BIC(exp_nlme),
142                     LogLik = logLik(exp_nlme)))
143     }
144
145     if(exists("logistic_nlme") && !is.null(logistic_nlme)) {
146         model_comparison <- rbind(model_comparison,
147                                   data.frame(Model = "Logistic模型",
148                                               AIC = AIC(logistic_nlme),
149                                               BIC = BIC(logistic_nlme),
150                                               LogLik = logLik(logistic_nlme)))
151     }
152
153     if(exists("gompertz_nlme") && !is.null(gompertz_nlme)) {
154         model_comparison <- rbind(model_comparison,
155                                   data.frame(Model = "Gompertz模型",
156                                               AIC = AIC(gompertz_nlme),
157                                               BIC = BIC(gompertz_nlme),
158                                               LogLik = logLik(gompertz_nlme)))
159     }
160
161     if(exists("gamm_model") && !is.null(gamm_model)) {
162         model_comparison <- rbind(model_comparison,
163                                   data.frame(Model = "GAMM模型",
164                                               AIC = AIC(gamm_model),
165                                               BIC = BIC(gamm_model),
166                                               LogLik = logLik(gamm_model)))
167     }
168
169     # 显示模型比较结果
170     print("模型比较结果:")
171     print(model_comparison)
172
173     # 可视化拟合结果
174     # 创建预测函数
175     predict_nlme <- function(model, newdata) {
176         if(inherits(model, "nlme")) {
177             return(predict(model, newdata = newdata, level = 0))
178         } else if(inherits(model, "lmerMod")) {
179             return(predict(model, newdata = newdata, re.form = NA))
180         } else if(inherits(model, "bam")) {
181             return(predict(model, newdata = newdata, exclude = "s(Subject)"))
182         } else {
183             return(rep(NA, nrow(newdata)))
184         }
185     }
186
187     # 生成预测数据 - 修复: 使用平均BMI值
188     pred_data <- data.frame(
189         BMI = rep(mean(data$BMI), 100), # 使用平均BMI值
190         Gestational_days = seq(min(data$Gestational_days), max(data$Gestational_days), length.out = 100),
191         Subject = rep(levels(data$Subject)[1], 100) # 使用第一个受试者
192     )
193

```

```

194 # 为每个模型生成预测
195 pred_data$lm <- predict_nlme(lmm_model, pred_data)
196
197 if(exists("exp_nlme") && !is.null(exp_nlme)) {
198   pred_data$exp <- predict_nlme(exp_nlme, pred_data)
199 }
200
201 if(exists("logistic_nlme") && !is.null(logistic_nlme)) {
202   pred_data$logistic <- predict_nlme(logistic_nlme, pred_data)
203 }
204
205 if(exists("gompertz_nlme") && !is.null(gompertz_nlme)) {
206   pred_data$gompertz <- predict_nlme(gompertz_nlme, pred_data)
207 }
208
209 if(exists("gamm_model") && !is.null(gamm_model)) {
210   pred_data$gamm <- predict_nlme(gamm_model, pred_data)
211 }
212
213 # 创建每个模型的拟合图
214 plots <- list()
215
216 # 线性混合模型
217 plots[[1]] <- ggplot() +
218   geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha = 0.6)
219   +
220   geom_line(data = pred_data, aes(x = Gestational_days, y = lm, color = "总体拟合"), size = 1) +
221   theme_minimal() +
222   labs(title = "线性混合模型拟合", x = "孕周(天)", y = "Y染色体浓度")
223
224 # 指数模型
225 if(exists("exp_nlme") && !is.null(exp_nlme)) {
226   plots[[2]] <- ggplot() +
227     geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha =
228       0.6) +
229     geom_line(data = pred_data, aes(x = Gestational_days, y = exp, color = "总体拟合"), size = 1) +
230     theme_minimal() +
231     labs(title = "指数模型拟合", x = "孕周(天)", y = "Y染色体浓度")
232 }
233
234 # Logistic模型
235 if(exists("logistic_nlme") && !is.null(logistic_nlme)) {
236   plots[[3]] <- ggplot() +
237     geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha =
238       0.6) +
239     geom_line(data = pred_data, aes(x = Gestational_days, y = logistic, color = "总体拟合"), size = 1) +
240     theme_minimal() +
241     labs(title = "Logistic模型拟合", x = "孕周(天)", y = "Y染色体浓度")
242 }
243
244 # Gompertz模型
245 if(exists("gompertz_nlme") && !is.null(gompertz_nlme)) {
246   plots[[4]] <- ggplot() +
247     geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha =
248       0.6) +

```

```

245   geom_line(data = pred_data, aes(x = Gestational_days, y = gompertz, color = "总体拟合"), size = 1) +
246   theme_minimal() +
247   labs(title = "Gompertz模型拟合", x = "孕周(天)", y = "Y染色体浓度")
248 }
249
250 # GAMM模型
251 if(exists("gamm_model") && !is.null(gamm_model)) {
252   plots[[5]] <- ggplot() +
253     geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha =
254       0.6) +
255     geom_line(data = pred_data, aes(x = Gestational_days, y = gamm, color = "总体拟合"), size = 1) +
256     theme_minimal() +
257     labs(title = "GAMM模型拟合", x = "孕周(天)", y = "Y染色体浓度")
258 }
259
260 # 保存所有图表
261 for(i in seq_along(plots)) {
262   ggsave(paste0("model_fit_", i, ".png"), plots[[i]], width = 10, height = 6, dpi = 300)
263 }
264
265 # 创建模型比较图
266 model_comp_plot <- ggplot(model_comparison, aes(x = reorder(Model, AIC), y = AIC, fill = Model)) +
267   geom_bar(stat = "identity") +
268   theme_minimal() +
269   labs(title = "模型比较 - AIC值", x = "模型", y = "AIC值") +
270   theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
271   scale_fill_brewer(palette = "Set2")
272
273 ggsave("model_comparison_aic.png", model_comp_plot, width = 10, height = 6, dpi = 300)
274
275 # 残差分析
276 residual_plots <- list()
277
278 # 线性混合模型残差
279 residual_plots[[1]] <- ggplot(data, aes(x = fitted(lmm_model), y = resid(lmm_model))) +
280   geom_point(alpha = 0.6) +
281   geom_hline(yintercept = 0, linetype = "dashed") +
282   theme_minimal() +
283   labs(title = "线性混合模型残差图", x = "拟合值", y = "残差")
284
285 # 指数模型残差
286 if(exists("exp_nlme") && !is.null(exp_nlme)) {
287   residual_plots[[2]] <- ggplot(data, aes(x = fitted(exp_nlme), y = resid(exp_nlme))) +
288     geom_point(alpha = 0.6) +
289     geom_hline(yintercept = 0, linetype = "dashed") +
290     theme_minimal() +
291     labs(title = "指数模型残差图", x = "拟合值", y = "残差")
292 }
293
294 # GAMM模型残差
295 if(exists("gamm_model") && !is.null(gamm_model)) {
296   residual_plots[[3]] <- ggplot(data, aes(x = fitted(gamm_model), y = resid(gamm_model))) +
297     geom_point(alpha = 0.6) +
298     geom_hline(yintercept = 0, linetype = "dashed") +
299     theme_minimal() +

```

```

299     labs(title = "GAMM模型残差图", x = "拟合值", y = "残差")
300 }
301
302 # 保存残差图
303 for(i in seq_along(residual_plots)) {
304     ggsave(paste0("residual_plot_", i, ".png"), residual_plots[[i]], width = 10, height = 6, dpi = 300)
305 }
306
307 # 输出模型摘要
308 sink("model_summaries.txt")
309 cat("\n线性混合模型摘要:\n")
310 print(summary(lmm_model))
311
312 if(exists("exp_nlme") && !is.null(exp_nlme)) {
313     cat("\n指数模型摘要:\n")
314     print(summary(exp_nlme))
315 }
316
317 if(exists("logistic_nlme") && !is.null(logistic_nlme)) {
318     cat("\nLogistic模型摘要:\n")
319     print(summary(logistic_nlme))
320 }
321
322 if(exists("gompertz_nlme") && !is.null(gompertz_nlme)) {
323     cat("\nGompertz模型摘要:\n")
324     print(summary(gompertz_nlme))
325 }
326
327 if(exists("gamm_model") && !is.null(gamm_model)) {
328     cat("\nGAMM模型摘要:\n")
329     print(summary(gamm_model))
330 }
331
332 cat("\n模型比较:\n")
333 print(model_comparison)
334 sink()
335
336 # 显示最佳模型
337 if(nrow(model_comparison) > 0) {
338     best_model_idx <- which.min(model_comparison$AIC)
339     cat("根据AIC准则, 最佳模型是:", model_comparison$Model[best_model_idx], "\n")
340
341 # 绘制最佳模型拟合图
342 if(model_comparison$Model[best_model_idx] == "线性混合模型") {
343     best_plot <- plots[[1]]
344 } else if(model_comparison$Model[best_model_idx] == "指数模型") {
345     best_plot <- plots[[2]]
346 } else if(model_comparison$Model[best_model_idx] == "Logistic模型") {
347     best_plot <- plots[[3]]
348 } else if(model_comparison$Model[best_model_idx] == "Gompertz模型") {
349     best_plot <- plots[[4]]
350 } else if(model_comparison$Model[best_model_idx] == "GAMM模型") {
351     best_plot <- plots[[5]]
352 }
353

```

```

354     ggsave("best_model_fit.png", best_plot, width = 10, height = 6, dpi = 300)
355 }
356
357 # 保存工作空间
358 save.image("mixed_model_analysis.RData")
359
360 cat("分析完成！所有结果已保存到当前工作目录。\\n")

```

Listing 2: C(2)(权重计算).py

```

1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy import stats
5
6  plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
7  plt.rcParams["axes.unicode_minus"] = False
8
9
10 def calculate_ahp_weights(matrix):
11
12     n = matrix.shape[0]
13
14     row_geom_mean = np.prod(matrix, axis=1) ** (1 / n)
15
16     weights = row_geom_mean / np.sum(row_geom_mean)
17
18     aw = np.dot(matrix, weights)
19     lambda_max = np.mean(aw / weights)
20
21     ci = (lambda_max - n) / (n - 1)
22
23     ri_table = {
24         1: 0,
25         2: 0,
26         3: 0.58,
27         4: 0.90,
28         5: 1.12,
29         6: 1.24,
30         7: 1.32,
31         8: 1.41,
32         9: 1.45,
33         10: 1.49,
34     }
35     ri = ri_table.get(n, 1.51)
36
37     cr = ci / ri if ri != 0 else 0
38
39     return weights, cr
40
41
42 def generate_stacked_bmi_mixed_charts_with_ahp_weights():
43     judgment_matrix = np.array([[1, 4, 6], [1 / 4, 1, 2], [1 / 6, 1 / 2, 1]])
44
45     category_order = ["always_can", "middle", "cannot"]

```

```

46
47     ahp_weights, cr = calculate_ahp_weights(judgment_matrix)
48
49     print("--- AHP 权重计算结果 ---")
50     print(f"判断矩阵:\n{judgment_matrix}\n")
51     print("计算出的权重向量:")
52     for i, category in enumerate(category_order):
53         print(f"    - {category}: {ahp_weights[i]:.4f}")
54
55     print(f"\n一致性指标 CI: {cr*0.58:.4f}")
56     print(f"一致性比率 CR: {cr:.4f}")
57     if cr < 0.1:
58         print(" -> 判断矩阵具有满意的一致性 (CR < 0.1)。")
59     else:
60         print(" -> 警告: 判断矩阵的一致性较差 (CR >= 0.1), 建议调整比较值。")
61     print("-" * 28 + "\n")
62
63     objective_weights_dict = {
64         category: weight for category, weight in zip(category_order, ahp_weights)
65     }
66
67     df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
68     df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
69     df_always_can_test = pd.read_excel(
70         "./python_code/bmi_Y_always_can_test_result.xlsx"
71     )
72
73     df_middle["category"] = "middle"
74     df_cannot_test["category"] = "cannot"
75     df_always_can_test["category"] = "always_can"
76
77     df_all = pd.concat(
78         [df_middle, df_cannot_test, df_always_can_test], ignore_index=True
79     )
80
81     df_all["days_raw"] = 0
82     df_all.loc[df_all["category"] == "cannot", "days_raw"] = df_all["最晚不达标天数"]
83     df_all.loc[df_all["category"] == "middle", "days_raw"] = df_all["预测达标天数"]
84     df_all.loc[df_all["category"] == "always_can", "days_raw"] = df_all["最早达标天数"]
85
86     def categorize_bmi(bmi):
87         if bmi < 30.17:
88             return "<30.17"
89         elif 30.17 <= bmi < 32.25:
90             return "30.17-32.25"
91         elif 32.25 <= bmi < 34.70:
92             return "32.25-34.70"
93         elif 34.70 <= bmi < 37.11:
94             return "34.70-37.11"
95         else:
96             return ">37.11"
97
98     df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)
99
100    df_all["weight"] = df_all["category"].map(objective_weights_dict)

```

```

101
102     bmi_categories = ["<30.17", "30.17-32.25", "32.25-34.70", "34.70-37.11", ">37.11"]
103
104     stacking_order = ["cannot", "middle", "always_can"]
105     colors = {"cannot": "lightcoral", "middle": "goldenrod", "always_can": "seagreen"}
106     labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}
107
108     for bmi_cat in bmi_categories:
109         df_bmi = df_all[df_all["bmi_category"] == bmi_cat]
110
111         if df_bmi.empty:
112             continue
113
114         fig, ax1 = plt.subplots(figsize=(12, 8))
115
116         all_days_raw = df_bmi["days_raw"].tolist()
117
118         stacked_data, stacked_labels, stacked_colors = [], [], []
119         for category in stacking_order:
120             days = df_bmi[df_bmi["category"] == category]["days_raw"].tolist()
121             if days:
122                 stacked_data.append(days)
123                 stacked_labels.append(labels[category])
124                 stacked_colors.append(colors[category])
125
126         if stacked_data:
127             data_range = max(all_days_raw) - min(all_days_raw) if all_days_raw else 0
128             bins_count = (
129                 min(100, max(50, int(data_range / 3))) if data_range > 0 else 50
130             )
131             ax1.hist(
132                 stacked_data,
133                 bins=bins_count,
134                 stacked=True,
135                 color=stacked_colors,
136                 label=stacked_labels,
137                 alpha=0.95,
138                 edgecolor="black",
139                 linewidth=0.1,
140             )
141
142             ax1.set_xlabel("天数", fontsize=12)
143             ax1.set_ylabel("人数", fontsize=12)
144             ax1.set_title(f"BMI 区间 {bmi_cat} 分布 (AHP 业务逻辑权重)", fontsize=14)
145             ax1.legend(loc="upper left", fontsize=10)
146             ax1.grid(True, alpha=0.3)
147
148             if len(all_days_raw) > 1:
149                 ax2 = ax1.twinx()
150                 days_for_kde = df_bmi["days_raw"].to_numpy()
151                 weights_for_kde = df_bmi["weight"].to_numpy()
152
153                 try:
154                     weighted_kde = stats.gaussian_kde(days_for_kde, weights=weights_for_kde)
155                     x_range = np.linspace(min(all_days_raw), max(all_days_raw), 1000)

```

```

156         kde_values = weighted_kde(x_range)
157
158         ax2.plot(
159             x_range,
160             kde_values,
161             color="dodgerblue",
162             linewidth=2,
163             linestyle="--",
164             label="AHP 加权概率密度",
165         )
166         ax2.set_ylabel("加权概率密度", fontsize=12)
167         ax2.legend(loc="upper right", fontsize=10)
168
169         cumulative_prob = np.cumsum(kde_values) * (x_range[1] - x_range[0])
170         prob_levels = [0.85, 0.90, 0.95]
171         for prob in prob_levels:
172             try:
173                 index = np.where(cumulative_prob >= prob)[0][0]
174                 day_value = x_range[index]
175                 ax2.axvline(
176                     x=day_value, color="purple", linestyle="--", linewidth=1
177                 )
178                 ax2.text(
179                     day_value + 0.5,
180                     max(kde_values) * (1 - prob),
181                     f"{int(prob*100)}% -> {day_value:.1f}天",
182                     color="purple",
183                     rotation=90,
184                 )
185             except IndexError:
186                 pass
187
188         except Exception as e:
189             print(f"BMI 区间 {bmi_cat} 计算加权概率密度时出错: {e}")
190
191         plt.tight_layout()
192         filename = f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt").replace("-", "_")}
193             _AHP_weighted_dist.png'
194         plt.savefig(filename, dpi=300, bbox_inches="tight")
195         plt.close()
196
197 generate_stacked_bmi_mixed_charts_with_ahp_weights()

```

Listing 3: C(2)(生存函数).py

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from lifelines import KaplanMeierFitter
5 from scipy.optimize import minimize_scalar
6
7 plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
8 plt.rcParams["axes.unicode_minus"] = False
9

```



```

10
11 def analyze_with_dynamic_utility():
12     def get_rearly(t):
13         return 2.0 * np.exp(-t / 50)
14
15     def get_rlate(t):
16         if t < 84:
17             return 0.1
18         elif t <= 189:
19             normalized_t = (t - 84) / (189 - 84)
20             return 0.1 + 0.9 * (normalized_t**2)
21         else:
22             return 1.0
23
24     def calculate_utility(t, survival_func):
25         if t < 0:
26             return float("inf")
27         closest_time_idx = np.abs(survival_func.index - t).argmin()
28         actual_time = survival_func.index[closest_time_idx]
29         p_t = 1 - survival_func.loc[actual_time, "s_t"]
30         return (1 - p_t) * get_rearly(t) + p_t * get_rlate(t)
31
32     df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
33     df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
34     df_always_can_test = pd.read_excel(
35         "./python_code/bmi_Y_always_can_test_result.xlsx"
36     )
37
38     df_all = pd.concat(
39         [
40             df_middle.assign(category="middle"),
41             df_cannot_test.assign(category="cannot"),
42             df_always_can_test.assign(category="always_can"),
43         ],
44         ignore_index=True,
45     )
46
47     df_all["duration"] = 0.0
48     df_all["event_observed"] = 0
49
50     df_all.loc[df_all["category"] == "cannot", "duration"] = df_all["最晚不达标天数"]
51     df_all.loc[df_all["category"] == "cannot", "event_observed"] = 0
52
53     df_all.loc[df_all["category"] == "middle", "duration"] = df_all["预测达标天数"]
54     df_all.loc[df_all["category"] == "middle", "event_observed"] = 1
55
56     df_all.loc[df_all["category"] == "always_can", "duration"] = 0.1
57     df_all.loc[df_all["category"] == "always_can", "event_observed"] = 1
58
59     def categorize_bmi(bmi):
60         if bmi < 30.26:
61             return "<30.26"
62         elif 30.26 <= bmi < 32.30:
63             return "30.26-32.30"
64         elif 32.30 <= bmi < 34.92:

```

```

65         return "32.30-34.92"
66     elif 34.92 <= bmi < 39.49:
67         return "34.92-39.49"
68     else:
69         return ">39.49"
70
71 df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)
72
73 bmi_categories = ["<30.26", "30.26-32.30", "32.30-34.92", "34.92-39.49", ">39.49"]
74
75 results_table = []
76
77 for bmi_cat in bmi_categories:
78     df_bmi = df_all[df_all["bmi_category"] == bmi_cat].copy()
79
80     if df_bmi.empty:
81         results_table.append(
82             {
83                 "BMI分组": bmi_cat,
84                 "最优时点(天)": "无数据",
85                 "最小效用值": "无数据",
86                 "风险水平": "无数据",
87                 "样本量": 0,
88             }
89         )
90         continue
91
92     print(f"--- 正在分析 BMI 区间: {bmi_cat} ---")
93
94     kmf = KaplanMeierFitter()
95     kmf.fit(durations=df_bmi["duration"], event_observed=df_bmi["event_observed"])
96
97     s_t = kmf.survival_function_.rename(columns={"KM_estimate": "s_t"})
98
99     def objective(t):
100         return calculate_utility(t, s_t)
101
102     result = minimize_scalar(
103         objective, bounds=(0, df_bmi["duration"].max()), method="bounded"
104     )
105
106     optimal_time = result.x
107     optimal_utility = result.fun
108     risk_level = 1 / optimal_utility if optimal_utility > 0 else float("inf")
109     sample_size = len(df_bmi)
110
111     print(f"\n最优预测时间点分析:")
112     print(f" - 最优时间点: {optimal_time:.1f} 天")
113     print(f" - 最小效用值: {optimal_utility:.4f}")
114     print(f" - 风险水平: {risk_level:.4f}")
115     print(f" - 样本量: {sample_size}")
116
117     plt.figure(figsize=(14, 8))
118     ax = plt.gca()
119

```

```

120 (1 - kmf.survival_function_).plot(
121     ax=ax,
122     label=f"累积达标函数 F(t) ({bmi_cat})",
123     color="seagreen",
124     linewidth=2.5,
125 )
126
127 time_points = np.linspace(0, df_bmi["duration"].max(), 200)
128 utilities = [calculate_utility(t, s_t) for t in time_points]
129 ax_twin = ax.twinx()
130 ax_twin.plot(
131     time_points, utilities, "--", color="red", label="效用函数 E(t)", alpha=0.6
132 )
133
134 ax_twin.axvline(x=optimal_time, color="purple", linestyle="--", linewidth=1)
135 ax_twin.text(
136     optimal_time + 2,
137     min(utilities) + 0.1,
138     f"最优时间点: {optimal_time:.1f}天\n效用值: {optimal_utility:.3f}",
139     color="purple",
140     fontsize=10,
141 )
142
143 ax.axvspan(0, 84, facecolor="green", alpha=0.15, label="早期阶段 (<84天)")
144 ax.axvspan(84, 189, facecolor="orange", alpha=0.15, label="过渡阶段 (84-189天)")
145 ax.axvspan(
146     189,
147     df_all["duration"].max() + 10,
148     facecolor="red",
149     alpha=0.15,
150     label="晚期阶段 (>189天)",
151 )
152
153 ax.set_title(f"BMI区间 {bmi_cat} 达标情况与效用分析", fontsize=16)
154 ax.set_xlabel("天数", fontsize=12)
155 ax.set_ylabel("累积达标比例", fontsize=12)
156 ax_twin.set_ylabel("效用值", fontsize=12)
157
158 lines1, labels1 = ax.get_legend_handles_labels()
159 lines2, labels2 = ax_twin.get_legend_handles_labels()
160 ax.legend(lines1 + lines2, labels1 + labels2, loc="upper left")
161
162 ax.grid(True, alpha=0.3)
163 ax.set_ylim(0, 1.05)
164 ax_twin.set_ylim(0, max(utilities) * 1.2)
165 ax.set_xlim(-5, df_all["duration"].max() + 10)
166
167 filename = f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt").replace("-", "_")}_utility_analysis.png'
168 plt.savefig(filename, dpi=300, bbox_inches="tight")
169 plt.close()
170
171 results_table.append(
172     {
173         "BMI分组": bmi_cat,

```

```

174         "最优时点(天)": f"{optimal_time:.1f}",
175         "最小效用值": f"{optimal_utility:.4f}",
176         "风险水平": f"{risk_level:.4f}",
177         "样本量": sample_size,
178     }
179 )
180
181 print(f"\n图表已保存至: {filename}")
182 print("\n" + "=" * 50 + "\n")
183
184 print("\n\n=== 各BMI分组NIPT时点计算结果 ===")
185 print("BMI分组\t\t最优时点(天)\t最小效用值\t风险水平\t样本量")
186 print("-" * 70)
187
188 for result in results_table:
189     print(
190         f"{result['BMI分组']}\t\t{result['最优时点(天)']}\t\t{result['最小效用值']}\t\t{result['风险水平']}\t\t{result['样本量']}"
191     )
192
193 results_df = pd.DataFrame(results_table)
194 results_df.to_excel("./python_code/NIPT_optimal_times_results.xlsx", index=False)
195 print(f"\n结果表格已保存至: ./python_code/NIPT_optimal_times_results.xlsx")
196
197 return results_df
198
199
200 results = analyze_with_dynamic_utility()

```

Listing 4: C(2)(正态分布检验).py

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.font_manager as fm
5 from matplotlib import rcParams
6 from scipy import stats
7 import re
8
9 plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
10 plt.rcParams["axes.unicode_minus"] = False
11
12
13 def categorize_bmi(bmi):
14     if bmi < 30:
15         return "<30"
16     elif 30 <= bmi < 32:
17         return "30-32"
18     elif 32 <= bmi < 34:
19         return "32-34"
20     elif 34 <= bmi < 36:
21         return "34-36"
22     else:
23         return ">36"
24

```

```

25
26 def normality_test_and_plot_with_qq():
27     df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
28     df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
29     df_always_can_test = pd.read_excel(
30         "./python_code/bmi_Y_always_can_test_result.xlsx"
31     )
32
33     df_middle["bmi_category"] = df_middle["BMI"].apply(categorize_bmi)
34     df_cannot_test["bmi_category"] = df_cannot_test["BMI"].apply(categorize_bmi)
35     df_always_can_test["bmi_category"] = df_always_can_test["BMI"].apply(categorize_bmi)
36
37     bmi_categories = ["<30", "30-32", "32-34", "34-36", ">36"]
38
39     colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
40     labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}
41
42     for bmi_cat in bmi_categories:
43         df_cannot = df_cannot_test[df_cannot_test["bmi_category"] == bmi_cat]
44         df_middle_cat = df_middle[df_middle["bmi_category"] == bmi_cat]
45         df_always = df_always_can_test[df_always_can_test["bmi_category"] == bmi_cat]
46
47         days_cannot = (
48             df_cannot["最晚不达标天数"].tolist() if not df_cannot.empty else []
49         )
50         days_middle = (
51             df_middle_cat["预测达标天数"].tolist() if not df_middle_cat.empty else []
52         )
53         days_always = df_always["最早达标天数"].tolist() if not df_always.empty else []
54
55         all_days = days_cannot + days_middle + days_always
56
57         if not all_days:
58             print(f"警告: BMI 区间 {bmi_cat} 没有数据")
59             continue
60
61         data = np.array(all_days)
62
63         print(f"\n=== BMI 区间 {bmi_cat} 的正态分布检验 ===")
64         print(f"样本数量: {len(data)}")
65
66         if len(data) <= 5000:
67             try:
68                 shapiro_stat, shapiro_p = stats.shapiro(data)
69                 print(
70                     f"Shapiro-Wilk 检验: 统计量={shapiro_stat:.6f}, p 值={shapiro_p:.6f}"
71                 )
72                 if shapiro_p > 0.05:
73                     print(" 结论: 数据符合正态分布 (p > 0.05)")
74                 else:
75                     print(" 结论: 数据不符合正态分布 (p <= 0.05)")
76             except Exception as e:
77                 print(f"Shapiro-Wilk 检验失败: {e}")
78         else:
79             print("Shapiro-Wilk 检验: 样本量过大(>5000), 跳过该检验")

```

```

80
81     try:
82         jb_stat, jb_p = stats.jarque_bera(data)
83         print(f"Jarque-Bera 检验: 统计量={jb_stat:.6f}, p值={jb_p:.6f}")
84         if jb_p > 0.05:
85             print(" 结论: 数据符合正态分布 (p > 0.05)")
86         else:
87             print(" 结论: 数据不符合正态分布 (p <= 0.05)")
88     except Exception as e:
89         print(f"Jarque-Bera 检验失败: {e}")
90
91     mean = np.mean(data)
92     std = np.std(data)
93     print(f"数据均值: {mean:.2f}, 标准差: {std:.2f}")
94
95     plt.figure(figsize=(12, 8))
96
97     data_to_plot = []
98     plot_labels = []
99     plot_colors = []
100
101     if days_cannot:
102         data_to_plot.append(days_cannot)
103         plot_labels.append(labels["cannot"])
104         plot_colors.append(colors["cannot"])
105
106     if days_middle:
107         data_to_plot.append(days_middle)
108         plot_labels.append(labels["middle"])
109         plot_colors.append(colors["middle"])
110
111     if days_always:
112         data_to_plot.append(days_always)
113         plot_labels.append(labels["always_can"])
114         plot_colors.append(colors["always_can"])
115
116     if data_to_plot:
117         n, bins, patches = plt.hist(
118             data_to_plot,
119             bins=20,
120             stacked=True,
121             color=plot_colors,
122             label=plot_labels,
123             alpha=0.7,
124             edgecolor="black",
125             linewidth=0.3,
126         )
127
128     is_normal = False
129     if len(data) <= 5000 and "shapiro_p" in locals() and shapiro_p > 0.05:
130         is_normal = True
131     elif len(data) > 5000 and "jb_p" in locals() and jb_p > 0.05:
132         is_normal = True
133     elif (
134         len(data) <= 5000

```

```

135         and "shapiro_p" not in locals()
136         and "jb_p" in locals()
137         and jb_p > 0.05
138     ):
139         is_normal = True
140
141     if is_normal:
142         x = np.linspace(min(data), max(data), 1000)
143         y = stats.norm.pdf(x, mean, std)
144         bin_width = bins[1] - bins[0]
145         y_scaled = y * len(data) * bin_width
146
147         plt.plot(
148             x,
149             y_scaled,
150             "b-",
151             linewidth=2,
152             label=f"正态分布拟合 (={mean:.1f}, =std:.1f)",
153         )
154
155         print(" 已在图中绘制正态分布拟合曲线")
156     else:
157         print(" 数据不符合正态分布，未绘制正态分布曲线")
158
159     plt.xlabel("天数", fontsize=14, fontweight="bold")
160     plt.ylabel("人数", fontsize=14, fontweight="bold")
161     plt.title(
162         f"BMI区间 {bmi_cat} 的孕妇Y染色体达标情况分布及正态分布检验",
163         fontsize=16,
164         fontweight="bold",
165     )
166     plt.legend(fontsize=12)
167     plt.grid(True, alpha=0.3)
168
169     plt.xticks(fontsize=12)
170     plt.yticks(fontsize=12)
171
172     plt.tight_layout()
173     normal_suffix = "_normal" if is_normal else "_non_normal"
174     plt.savefig(
175         f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_stacked_with_normal_test'
176         f'{normal_suffix}.png',
177         dpi=300,
178         bbox_inches="tight",
179     )
180     plt.close()
181
182     plt.figure(figsize=(10, 8))
183
184     stats.probplot(data, dist="norm", plot=plt)
185     plt.title(f"BMI区间 {bmi_cat} 的Q-Q图", fontsize=16, fontweight="bold")
186     plt.xlabel("理论分位数", fontsize=14)
187     plt.ylabel("样本分位数", fontsize=14)
188     plt.grid(True, alpha=0.3)

```

```

189     plt.xticks(fontsize=12)
190     plt.yticks(fontsize=12)
191
192     theoretical_quantiles, sample_quantiles = stats.probplot(data, dist="norm")
193     slope, intercept, r_value, p_value, std_err = stats.linregress(
194         theoretical_quantiles[0], theoretical_quantiles[1]
195     )
196     r_squared = r_value**2
197
198     plt.text(
199         0.05,
200         0.95,
201         f"R² = {r_squared:.4f}",
202         transform=plt.gca().transAxes,
203         fontsize=12,
204         verticalalignment="top",
205         bbox=dict(boxstyle="round", facecolor="white", alpha=0.8),
206     )
207
208     plt.tight_layout()
209     plt.savefig(
210         f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_qq_plot.png',
211         dpi=300,
212         bbox_inches="tight",
213     )
214     plt.close()
215
216     print(f"  已生成Q-Q图 (R² = {r_squared:.4f})")
217
218     print(f"BMI区间 {bmi_cat} 详细统计:")
219     print(f"  不能达标: {len(df_cannot)} 人")
220     print(f"  中间达标: {len(df_middle_cat)} 人")
221     print(f"  始终达标: {len(df_always)} 人")
222     print(f"  总计: {len(data)} 人")
223
224
225 if __name__ == "__main__":
226     normality_test_and_plot_with_qq()
227
228     print("\n正态分布检验、直方图和Q-Q图生成完成!")

```

Listing 5: C(2) 测试误差.py

```

1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from lifelines import KaplanMeierFitter
5  from scipy.optimize import minimize_scalar
6  from scipy.stats import norm
7  import re
8
9  plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
10 plt.rcParams["axes.unicode_minus"] = False
11
12

```



```

13 def analyze_with_error_simulation(num_simulations=100, error_std=0.01):
14     # 定义效用函数 (与之前相同)
15     def get_rearly(t):
16         return 2.0 * np.exp(-t / 50)
17
18     def get_rlate(t):
19         if t < 84:
20             return 0.1
21         elif t <= 189:
22             normalized_t = (t - 84) / (189 - 84)
23             return 0.1 + 0.9 * (normalized_t**2)
24         else:
25             return 1.0
26
27     def calculate_utility(t, survival_func):
28         if t < 0:
29             return float("inf")
30         closest_time_idx = np.abs(survival_func.index - t).argmin()
31         actual_time = survival_func.index[closest_time_idx]
32         p_t = 1 - survival_func.loc[actual_time, "s_t"]
33         return (1 - p_t) * get_rearly(t) + p_t * get_rlate(t)
34
35     # 辅助函数: 清理文件名中的非法字符
36     def clean_filename(name):
37         # 替换文件名字符串中的非法字符
38         return re.sub(r'[\<>:"/\|?*]', "_", name)
39
40     # 加载数据
41     df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
42     df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
43     df_always_can_test = pd.read_excel(
44         "./python_code/bmi_Y_always_can_test_result.xlsx"
45     )
46
47     df_all = pd.concat(
48         [
49             df_middle.assign(category="middle"),
50             df_cannot_test.assign(category="cannot"),
51             df_always_can_test.assign(category="always_can"),
52         ],
53         ignore_index=True,
54     )
55
56     df_all["duration"] = 0.0
57     df_all["event_observed"] = 0
58
59     df_all.loc[df_all["category"] == "cannot", "duration"] = df_all["最晚不达标天数"]
60     df_all.loc[df_all["category"] == "cannot", "event_observed"] = 0
61
62     df_all.loc[df_all["category"] == "middle", "duration"] = df_all["预测达标天数"]
63     df_all.loc[df_all["category"] == "middle", "event_observed"] = 1
64
65     df_all.loc[df_all["category"] == "always_can", "duration"] = 0.1
66     df_all.loc[df_all["category"] == "always_can", "event_observed"] = 1
67

```

```

68 def categorize_bmi(bmi):
69     if bmi < 30.26:
70         return "<30.26"
71     elif 30.26 <= bmi < 32.30:
72         return "30.26-32.30"
73     elif 32.30 <= bmi < 34.92:
74         return "32.30-34.92"
75     elif 34.92 <= bmi < 39.49:
76         return "34.92-39.49"
77     else:
78         return ">39.49"
79
80 df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)
81 bmi_categories = ["<30.26", "30.26-32.30", "32.30-34.92", "34.92-39.49", ">39.49"]
82
83 results = {
84     bmi_cat: {"times": [], "utilities": [], "risks": []}
85     for bmi_cat in bmi_categories
86 }
87
88 original_results = {}
89 for bmi_cat in bmi_categories:
90     df_bmi = df_all[df_all["bmi_category"] == bmi_cat].copy()
91     if df_bmi.empty:
92         continue
93
94     kmf = KaplanMeierFitter()
95     kmf.fit(durations=df_bmi["duration"], event_observed=df_bmi["event_observed"])
96     s_t = kmf.survival_function_.rename(columns={"KM_estimate": "s_t"})
97
98     def objective(t):
99         return calculate_utility(t, s_t)
100
101     result = minimize_scalar(
102         objective, bounds=(0, df_bmi["duration"].max()), method="bounded"
103     )
104
105     original_time = result.x
106     original_utility = result.fun
107     original_risk = 1 / original_utility if original_utility > 0 else float("inf")
108
109     original_results[bmi_cat] = {
110         "time": original_time,
111         "utility": original_utility,
112         "risk": original_risk,
113     }
114
115     print(f"BMI 区间 {bmi_cat} 原始结果:")
116     print(f"    最优时点: {original_time:.2f} 天")
117     print(f"    最小效用值: {original_utility:.4f}")
118     print(f"    风险水平: {original_risk:.4f}")
119
120 # 开始模拟
121 for simulation in range(num_simulations):
122     print(f"正在进行第 {simulation+1} 次模拟...")

```

```

123     df_simulated = df_all.copy()
124     np.random.seed(simulation)
125
126     mask_middle = df_simulated["category"] == "middle"
127     mask_always = df_simulated["category"] == "always_can"
128
129
130     if sum(mask_middle) > 0:
131         error_middle = np.random.normal(
132             0,
133             error_std * df_simulated.loc[mask_middle, "预测达标天数"].mean(),
134             size=sum(mask_middle),
135         )
136         df_simulated.loc[mask_middle, "duration"] += error_middle
137
138     if sum(mask_always) > 0:
139         error_always = np.random.normal(
140             0,
141             error_std * df_simulated.loc[mask_always, "最早达标天数"].mean(),
142             size=sum(mask_always),
143         )
144         df_simulated.loc[mask_always, "duration"] += error_always
145
146     mask_cannot = df_simulated["category"] == "cannot"
147     if sum(mask_cannot) > 0:
148         error_cannot = np.random.normal(
149             0,
150             error_std * df_simulated.loc[mask_cannot, "最晚不达标天数"].mean(),
151             size=sum(mask_cannot),
152         )
153         df_simulated.loc[mask_cannot, "duration"] += error_cannot
154
155     df_simulated["duration"] = df_simulated["duration"].clip(lower=0.1)
156
157     for bmi_cat in bmi_categories:
158         df_bmi = df_simulated[df_simulated["bmi_category"] == bmi_cat].copy()
159         if df_bmi.empty:
160             continue
161
162         kmf = KaplanMeierFitter()
163         kmf.fit(
164             durations=df_bmi["duration"], event_observed=df_bmi["event_observed"]
165         )
166         s_t = kmf.survival_function_.rename(columns={"KM_estimate": "s_t"})
167
168         def objective(t):
169             return calculate_utility(t, s_t)
170
171         result = minimize_scalar(
172             objective, bounds=(0, df_bmi["duration"].max()), method="bounded"
173         )
174
175         optimal_time = result.x
176         optimal_utility = result.fun
177         risk = 1 / optimal_utility if optimal_utility > 0 else float("inf")

```

```

178         results[bmi_cat]["times"].append(optimal_time)
179         results[bmi_cat]["utilities"].append(optimal_utility)
180         results[bmi_cat]["risks"].append(risk)
181
182
183     print("\n=== 检测误差影响分析 ===")
184     for bmi_cat in bmi_categories:
185         if results[bmi_cat]["times"]:
186             times = np.array(results[bmi_cat]["times"])
187             utilities = np.array(results[bmi_cat]["utilities"])
188             risks = np.array(results[bmi_cat]["risks"])
189
190             mean_time = times.mean()
191             std_time = times.std()
192             time_confidence_interval = norm.interval(
193                 0.95, loc=mean_time, scale=std_time / np.sqrt(len(times))
194             )
195
196             mean_utility = utilities.mean()
197             std_utility = utilities.std()
198
199             mean_risk = risks.mean()
200             std_risk = risks.std()
201
202             original_time = original_results.get(bmi_cat, {}).get("time", 0)
203             original_utility = original_results.get(bmi_cat, {}).get("utility", 0)
204             original_risk = original_results.get(bmi_cat, {}).get("risk", 0)
205
206             print(f"BMI 区间 {bmi_cat}:")
207             print(f"  原始最优时点: {original_time:.2f} 天")
208             print(f"  模拟最优时点均值: {mean_time:.2f} 天")
209             print(f"  标准差: {std_time:.2f} 天")
210             print(
211                 f"  95%置信区间: ({time_confidence_interval[0]:.2f}, {time_confidence_interval[1]:.2f})"
212                 f"  天"
213             )
214             print(f"  原始最小效用值: {original_utility:.4f}")
215             print(f"  模拟最小效用值均值: {mean_utility:.4f}")
216             print(f"  效用值标准差: {std_utility:.4f}")
217             print(f"  原始风险水平: {original_risk:.4f}")
218             print(f"  模拟风险水平均值: {mean_risk:.4f}")
219             print(f"  风险水平标准差: {std_risk:.4f}")
220
221             plt.figure(figsize=(12, 8))
222
223             plt.subplot(2, 2, 1)
224             plt.hist(times, bins=20, alpha=0.7, edgecolor="black")
225             plt.axvline(
226                 mean_time,
227                 color="r",
228                 linestyle="--",
229                 label=f"模拟平均时点: {mean_time:.2f}",
230             )
231             plt.axvline(
232                 original_time,

```

```

232         color="b",
233         linestyle="--",
234         label=f"原始最优时点: {original_time:.2f}",
235     )
236     plt.axvline(
237         time_confidence_interval[0],
238         color="g",
239         linestyle=":",
240         label="95%置信区间",
241     )
242     plt.axvline(time_confidence_interval[1], color="g", linestyle=":")
243     plt.xlabel("最优NIPT时点 (天)")
244     plt.ylabel("频数")
245     plt.title(f"BMI区间 {bmi_cat} - 最优时点分布")
246     plt.legend()
247     plt.grid(True, alpha=0.3)
248
249     plt.subplot(2, 2, 2)
250     plt.hist(utilities, bins=20, alpha=0.7, edgecolor="black", color="orange")
251     plt.axvline(
252         mean_utility,
253         color="r",
254         linestyle="--",
255         label=f"模拟平均效用值: {mean_utility:.4f}",
256     )
257     plt.axvline(
258         original_utility,
259         color="b",
260         linestyle="--",
261         label=f"原始最小效用值: {original_utility:.4f}",
262     )
263     plt.xlabel("最小效用值")
264     plt.ylabel("频数")
265     plt.title(f"BMI区间 {bmi_cat} - 最小效用值分布")
266     plt.legend()
267     plt.grid(True, alpha=0.3)
268
269     plt.subplot(2, 2, 3)
270     plt.scatter(times, utilities, alpha=0.6)
271     plt.axvline(original_time, color="b", linestyle="--", label="原始最优时点")
272     plt.axhline(
273         original_utility, color="b", linestyle="--", label="原始最小效用值"
274     )
275     plt.xlabel("最优时点 (天)")
276     plt.ylabel("最小效用值")
277     plt.title(f"BMI区间 {bmi_cat} - 时点与效用值关系")
278     plt.legend()
279     plt.grid(True, alpha=0.3)
280
281     plt.subplot(2, 2, 4)
282     plt.hist(risks, bins=20, alpha=0.7, edgecolor="black", color="red")
283     plt.axvline(
284         mean_risk,
285         color="r",
286         linestyle="--",

```

```

287         label=f"模拟平均风险: {mean_risk:.4f}",
288     )
289     plt.axvline(
290         original_risk,
291         color="b",
292         linestyle="--",
293         label=f"原始风险: {original_risk:.4f}",
294     )
295     plt.xlabel("风险水平")
296     plt.ylabel("频数")
297     plt.title(f"BMI 区间 {bmi_cat} - 风险水平分布")
298     plt.legend()
299     plt.grid(True, alpha=0.3)
300
301     plt.tight_layout()
302
303     safe_bmi_cat = clean_filename(bmi_cat)
304     filename = f"./python_code/error_analysis_BMI_{safe_bmi_cat}.png"
305     plt.savefig(filename, dpi=300, bbox_inches="tight")
306     plt.close()
307
308     data_df = pd.DataFrame({"time": times, "utility": utilities, "risk": risks})
309     csv_filename = f"./python_code/error_analysis_BMI_{safe_bmi_cat}.csv"
310     data_df.to_csv(csv_filename, index=False)
311
312     print(f" 图表已保存至: {filename}")
313     print(f" 数据已保存至: {csv_filename}")
314 else:
315     print(f"BMI 区间 {bmi_cat}: 无数据")
316
317
318 analyze_with_error_simulation(num_simulations=100, error_std=0.05)

```

Listing 6: C(2) 分类画图.py

```

1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.font_manager as fm
5  from matplotlib import rcParams
6  from scipy import stats
7  import re
8
9
10 plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
11 plt.rcParams["axes.unicode_minus"] = False
12
13
14 def best_bmi_Y():
15     def weeks_to_days(weeks_str):
16         match = re.match(r"(\d+)[wW](?:\+(\d+))?", str(weeks_str), re.IGNORECASE)
17         if match:
18             weeks = int(match.group(1))
19             days = int(match.group(2)) if match.group(2) else 0
20             return weeks * 7 + days

```

```

21     else:
22         print(f"无法解析孕周格式: {weeks_str}")
23         return None
24
25 df = pd.read_excel("./python_code/附件.xlsx", sheet_name=0)
26 df = df[["孕妇代码", "检测孕周", "孕妇BMI", "Y染色体浓度"]]
27 bmi_avg = df.groupby("孕妇代码")["孕妇BMI"].mean()
28 df["孕妇平均BMI"] = df["孕妇代码"].map(bmi_avg)
29 df["检测孕周_天数"] = df["检测孕周"].apply(weeks_to_days)
30
31 # print(df)
32 code_col = "孕妇代码"
33 day_col = "检测孕周_天数"
34 bmi_col = "孕妇平均BMI"
35 y_col = "Y染色体浓度"
36 # df_can_test_codes = df[df[y_col] >= 0.04][code_col].unique() # not left
37 # df_cannot_test_codes = df[~df[code_col].isin(df_can_test_codes)][code_col].unique() # left
38 # df_codes_cannot_test_before_codes = df[df[y_col] < 0.04][code_col].unique() # not right
39 # df_always_can_test_codes = df[~df[code_col].isin(df_codes_cannot_test_before_codes)][code_col].
    unique() # right
40 # df_middle_codes = np.setdiff1d(df_can_test_codes, df_always_can_test_codes)
41 df_codes = df[code_col].unique()
42
43 result_middle = []
44 result_cannot_test = []
45 result_always_can_test = []
46 for code in df_codes:
47     data = df[df[code_col] == code].sort_values(by=day_col)
48     mean_bmi = data[bmi_col].mean()
49     first_ok_idx = None
50     for idx, row in data.iterrows():
51         if row[y_col] >= 0.04:
52             first_ok_idx = idx
53             break
54
55     if first_ok_idx is None:
56         result_cannot_test.append(
57             {code_col: code, "BMI": mean_bmi, "最晚不达标天数": data[day_col].max()}
58         )
59         continue
60     else:
61         later_data = data.loc[first_ok_idx:]
62         if (later_data[y_col] < 0.04).any():
63             continue
64         else:
65             if first_ok_idx == data.index[0]:
66                 result_always_can_test.append(
67                     {
68                         code_col: code,
69                         "BMI": mean_bmi,
70                         "最早达标天数": data[day_col].min(),
71                     }
72                 )
73             else:
74                 ok_row = data.loc[first_ok_idx]

```

```

75         prev_row = data.loc[
76             data.index[data.index.get_loc(first_ok_idx) - 1]
77         ]
78
79         w1 = abs(ok_row[y_col] - 0.04)
80         w2 = abs(prev_row[y_col] - 0.04)
81         predicted_day = (w2 * ok_row[day_col] + w1 * prev_row[day_col]) / (
82             w1 + w2
83         )
84         result_middle.append(
85             {code_col: code, "BMI": mean_bmi, "预测达标天数": predicted_day}
86         )
87
88     df_middle = pd.DataFrame(result_middle)
89     df_cannot_test = pd.DataFrame(result_cannot_test)
90     df_always_can_test = pd.DataFrame(result_always_can_test)
91
92     df_middle.to_excel("./python_code/bmi_Y_middle_result.xlsx", index=False)
93     df_cannot_test.to_excel("./python_code/bmi_Y_cannot_test_result.xlsx", index=False)
94     df_always_can_test.to_excel(
95         "./python_code/bmi_Y_always_can_test_result.xlsx", index=False
96     )
97
98
99 def generate_bmi_category_charts():
100     df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
101     df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
102     df_always_can_test = pd.read_excel(
103         "./python_code/bmi_Y_always_can_test_result.xlsx"
104     )
105
106     df_middle["category"] = "middle"
107     df_cannot_test["category"] = "cannot"
108     df_always_can_test["category"] = "always_can"
109
110     df_all = pd.concat(
111         [df_middle, df_cannot_test, df_always_can_test], ignore_index=True
112     )
113
114     def categorize_bmi(bmi):
115         if bmi < 30:
116             return "<30"
117         elif 30 <= bmi < 32:
118             return "30-32"
119         elif 32 <= bmi < 34:
120             return "32-34"
121         elif 34 <= bmi < 36:
122             return "34-36"
123         else:
124             return ">36"
125
126     df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)
127
128     bmi_categories = ["<30", "30-32", "32-34", "34-36", ">36"]
129

```



```

130 categories = ["cannot", "middle", "always_can"]
131 colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
132 labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}
133
134 for bmi_cat in bmi_categories:
135     df_bmi = df_all[df_all["bmi_category"] == bmi_cat]
136
137     if df_bmi.empty:
138         print(f"警告: BMI区间 {bmi_cat} 没有数据")
139         continue
140
141     plt.figure(figsize=(12, 8))
142
143     for category in categories:
144         df_category = df_bmi[df_bmi["category"] == category]
145
146         if not df_category.empty:
147             if category == "cannot":
148                 days = df_category["最晚不达标天数"]
149             elif category == "middle":
150                 days = df_category["预测达标天数"]
151             else:
152                 days = df_category["最早达标天数"]
153
154             plt.hist(
155                 days,
156                 bins=20,
157                 alpha=0.7,
158                 color=colors[category],
159                 label=labels[category],
160                 edgecolor="black",
161                 linewidth=0.5,
162             )
163
164             plt.xlabel("天数", fontsize=12)
165             plt.ylabel("人数", fontsize=12)
166             plt.title(f"BMI区间 {bmi_cat} 的孕妇Y染色体达标情况分布", fontsize=14)
167             plt.legend(fontsize=10)
168             plt.grid(True, alpha=0.3)
169
170             plt.savefig(
171                 f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_distribution.png',
172                 dpi=300,
173                 bbox_inches="tight",
174             )
175             plt.close()
176
177             print(f"BMI区间 {bmi_cat} 统计:")
178             for category in categories:
179                 count = len(df_bmi[df_bmi["category"] == category])
180                 print(f" {labels[category]}: {count} 人")
181             print()
182
183
184 def generate_bmi_mixed_charts():

```

```

185
186 df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
187 df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
188 df_always_can_test = pd.read_excel(
189     "./python_code/bmi_Y_always_can_test_result.xlsx"
190 )
191
192 df_middle["category"] = "middle"
193 df_cannot_test["category"] = "cannot"
194 df_always_can_test["category"] = "always_can"
195
196 df_all = pd.concat(
197     [df_middle, df_cannot_test, df_always_can_test], ignore_index=True
198 )
199
200 def categorize_bmi(bmi):
201     if bmi < 30:
202         return "<30"
203     elif 30 <= bmi < 32:
204         return "30-32"
205     elif 32 <= bmi < 34:
206         return "32-34"
207     elif 34 <= bmi < 36:
208         return "34-36"
209     else:
210         return ">36"
211
212 df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)
213
214 bmi_categories = ["<30", "30-32", "32-34", "34-36", ">36"]
215
216 categories = ["cannot", "middle", "always_can"]
217 colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
218 labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}
219
220 for bmi_cat in bmi_categories:
221     df_bmi = df_all[df_all["bmi_category"] == bmi_cat]
222
223     if df_bmi.empty:
224         print(f"警告: BMI 区间 {bmi_cat} 没有数据")
225         continue
226
227     fig, ax1 = plt.subplots(figsize=(12, 8))
228
229     all_days = []
230
231     for category in categories:
232         df_category = df_bmi[df_bmi["category"] == category]
233
234         if not df_category.empty:
235             if category == "cannot":
236                 days = df_category["最晚不达标天数"]
237             elif category == "middle":
238                 days = df_category["预测达标天数"]
239             else:

```

```

240         days = df_category["最早达标天数"]
241
242         ax1.hist(
243             days,
244             bins=20,
245             alpha=0.7,
246             color=colors[category],
247             label=labels[category],
248             edgecolor="black",
249             linewidth=0.5,
250         )
251
252         all_days.extend(days.tolist())
253
254     ax1.set_xlabel("天数", fontsize=12)
255     ax1.set_ylabel("人数", fontsize=12)
256     ax1.set_title(f"BMI 区间 {bmi_cat} 的孕妇Y染色体达标情况分布", fontsize=14)
257     ax1.legend(fontsize=10)
258     ax1.grid(True, alpha=0.3)
259
260     if len(all_days) > 1:
261         ax2 = ax1.twinx()
262
263         try:
264             kde = stats.gaussian_kde(all_days)
265             x_range = np.linspace(min(all_days), max(all_days), 1000)
266             kde_values = kde(x_range)
267
268             ax2.plot(
269                 x_range,
270                 kde_values,
271                 color="blue",
272                 linewidth=2,
273                 linestyle="--",
274                 label="总体概率密度",
275             )
276
277             ax2.set_ylabel("概率密度", fontsize=12)
278             ax2.legend(loc="upper right", fontsize=10)
279         except Exception as e:
280             print(f"BMI 区间 {bmi_cat} 计算概率密度时出错: {e}")
281
282     plt.tight_layout()
283
284     plt.savefig(
285         f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_distribution.png',
286         dpi=300,
287         bbox_inches="tight",
288     )
289     plt.close()
290
291     print(f"BMI 区间 {bmi_cat} 统计:")
292     total_count = len(df_bmi)
293     for category in categories:
294         count = len(df_bmi[df_bmi["category"] == category])

```

```

295         percentage = (count / total_count) * 100 if total_count > 0 else 0
296         print(f" {labels[category]}: {count} 人 ({percentage:.1f}%)")
297     print(f" 总计: {total_count} 人")
298     print()
299
300
301 def generate_stacked_bmi_charts_alternative():
302     df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
303     df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
304     df_always_can_test = pd.read_excel(
305         "./python_code/bmi_Y_always_can_test_result.xlsx"
306     )
307
308     def categorize_bmi(bmi):
309         if bmi < 30:
310             return "<30"
311         elif 30 <= bmi < 32:
312             return "30-32"
313         elif 32 <= bmi < 34:
314             return "32-34"
315         elif 34 <= bmi < 36:
316             return "34-36"
317         else:
318             return ">36"
319
320     df_middle["bmi_category"] = df_middle["BMI"].apply(categorize_bmi)
321     df_cannot_test["bmi_category"] = df_cannot_test["BMI"].apply(categorize_bmi)
322     df_always_can_test["bmi_category"] = df_always_can_test["BMI"].apply(categorize_bmi)
323
324     bmi_categories = ["<30", "30-32", "32-34", "34-36", ">36"]
325
326     colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
327     labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}
328     category_order = ["cannot", "middle", "always_can"]
329
330     for bmi_cat in bmi_categories:
331         df_cannot = df_cannot_test[df_cannot_test["bmi_category"] == bmi_cat]
332         df_middle_cat = df_middle[df_middle["bmi_category"] == bmi_cat]
333         df_always = df_always_can_test[df_always_can_test["bmi_category"] == bmi_cat]
334
335         if df_cannot.empty and df_middle_cat.empty and df_always.empty:
336             print(f"警告: BMI区间 {bmi_cat} 没有数据")
337             continue
338
339         plt.figure(figsize=(12, 8))
340
341         days_cannot = (
342             df_cannot["最晚不达标天数"].tolist() if not df_cannot.empty else []
343         )
344         days_middle = (
345             df_middle_cat["预测达标天数"].tolist() if not df_middle_cat.empty else []
346         )
347         days_always = df_always["最早达标天数"].tolist() if not df_always.empty else []
348
349         all_days = days_cannot + days_middle + days_always

```

```

350     if not all_days:
351         print(f"警告: BMI 区间 {bmi_cat} 没有有效数据")
352         continue
353
354     bins = np.linspace(min(all_days), max(all_days), 21)
355     data_to_plot = []
356     plot_labels = []
357     plot_colors = []
358
359     if days_cannot:
360         data_to_plot.append(days_cannot)
361         plot_labels.append(labels["cannot"])
362         plot_colors.append(colors["cannot"])
363
364     if days_middle:
365         data_to_plot.append(days_middle)
366         plot_labels.append(labels["middle"])
367         plot_colors.append(colors["middle"])
368
369     if days_always:
370         data_to_plot.append(days_always)
371         plot_labels.append(labels["always_can"])
372         plot_colors.append(colors["always_can"])
373
374     if data_to_plot:
375         plt.hist(
376             data_to_plot,
377             bins=bins,
378             stacked=True,
379             color=plot_colors,
380             label=plot_labels,
381             alpha=0.7,
382             edgecolor="black",
383             linewidth=0.3,
384         )
385
386     plt.xlabel("天数", fontsize=14, fontweight="bold")
387     plt.ylabel("人数", fontsize=14, fontweight="bold")
388     plt.title(
389         f"BMI 区间 {bmi_cat} 的孕妇Y染色体达标情况分布",
390         fontsize=16,
391         fontweight="bold",
392     )
393     plt.legend(fontsize=12)
394     plt.grid(True, alpha=0.3)
395
396     plt.xticks(fontsize=12)
397     plt.yticks(fontsize=12)
398
399     plt.tight_layout()
400     plt.savefig(
401         f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_stacked_v2.png',
402         dpi=300,
403         bbox_inches="tight",
404     )

```

```

405     plt.close()
406
407     print(f"BMI 区间 {bmi_cat} 统计:")
408     print(f"    不能达标: {len(df_cannot)} 人")
409     print(f"    中间达标: {len(df_middle_cat)} 人")
410     print(f"    始终达标: {len(df_always)} 人")
411     print(f"    总计: {len(df_cannot) + len(df_middle_cat) + len(df_always)} 人")
412     print()
413
414
415 def categorize_bmi(bmi):
416     """根据BMI值分类"""
417     if bmi < 30:
418         return "<30"
419     elif 30 <= bmi < 32:
420         return "30-32"
421     elif 32 <= bmi < 34:
422         return "32-34"
423     elif 34 <= bmi < 36:
424         return "34-36"
425     else:
426         return ">36"
427
428
429 def normality_test_and_plot():
430     df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
431     df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
432     df_always_can_test = pd.read_excel(
433         "./python_code/bmi_Y_always_can_test_result.xlsx"
434     )
435
436     df_middle["bmi_category"] = df_middle["BMI"].apply(categorize_bmi)
437     df_cannot_test["bmi_category"] = df_cannot_test["BMI"].apply(categorize_bmi)
438     df_always_can_test["bmi_category"] = df_always_can_test["BMI"].apply(categorize_bmi)
439
440     bmi_categories = ["<30", "30-32", "32-34", "34-36", ">36"]
441
442     colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
443     labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}
444
445     for bmi_cat in bmi_categories:
446         df_cannot = df_cannot_test[df_cannot_test["bmi_category"] == bmi_cat]
447         df_middle_cat = df_middle[df_middle["bmi_category"] == bmi_cat]
448         df_always = df_always_can_test[df_always_can_test["bmi_category"] == bmi_cat]
449
450         days_cannot = (
451             df_cannot["最早不达标天数"].tolist() if not df_cannot.empty else []
452         )
453         days_middle = (
454             df_middle_cat["预测达标天数"].tolist() if not df_middle_cat.empty else []
455         )
456         days_always = df_always["最早达标天数"].tolist() if not df_always.empty else []
457
458         all_days = days_cannot + days_middle + days_always
459

```

```

460     if not all_days:
461         print(f"警告: BMI区间 {bmi_cat} 没有数据")
462         continue
463
464     data = np.array(all_days)
465
466     print(f"\n=== BMI区间 {bmi_cat} 的正态分布检验 ===")
467     print(f"样本数量: {len(data)}")
468
469     if len(data) <= 5000:
470         try:
471             shapiro_stat, shapiro_p = stats.shapiro(data)
472             print(
473                 f"Shapiro-Wilk检验: 统计量={shapiro_stat:.6f}, p值={shapiro_p:.6f}"
474             )
475             if shapiro_p > 0.05:
476                 print(" 结论: 数据符合正态分布 (p > 0.05)")
477             else:
478                 print(" 结论: 数据不符合正态分布 (p <= 0.05)")
479         except Exception as e:
480             print(f"Shapiro-Wilk检验失败: {e}")
481     else:
482         print("Shapiro-Wilk检验: 样本量过大(>5000), 跳过该检验")
483
484     try:
485         jb_stat, jb_p = stats.jarque_bera(data)
486         print(f"Jarque-Bera检验: 统计量={jb_stat:.6f}, p值={jb_p:.6f}")
487         if jb_p > 0.05:
488             print(" 结论: 数据符合正态分布 (p > 0.05)")
489         else:
490             print(" 结论: 数据不符合正态分布 (p <= 0.05)")
491     except Exception as e:
492         print(f"Jarque-Bera检验失败: {e}")
493
494     mean = np.mean(data)
495     std = np.std(data)
496     print(f"数据均值: {mean:.2f}, 标准差: {std:.2f}")
497
498     plt.figure(figsize=(12, 8))
499
500     data_to_plot = []
501     plot_labels = []
502     plot_colors = []
503
504     if days_cannot:
505         data_to_plot.append(days_cannot)
506         plot_labels.append(labels["cannot"])
507         plot_colors.append(colors["cannot"])
508
509     if days_middle:
510         data_to_plot.append(days_middle)
511         plot_labels.append(labels["middle"])
512         plot_colors.append(colors["middle"])
513
514     if days_always:

```

```

515     data_to_plot.append(days_always)
516     plot_labels.append(labels["always_can"])
517     plot_colors.append(colors["always_can"])
518
519     if data_to_plot:
520         n, bins, patches = plt.hist(
521             data_to_plot,
522             bins=20,
523             stacked=True,
524             color=plot_colors,
525             label=plot_labels,
526             alpha=0.7,
527             edgecolor="black",
528             linewidth=0.3,
529         )
530
531     is_normal = False
532     if len(data) <= 5000 and "shapiro_p" in locals() and shapiro_p > 0.05:
533         is_normal = True
534     elif len(data) > 5000 and "jb_p" in locals() and jb_p > 0.05:
535         is_normal = True
536     elif (
537         len(data) <= 5000
538         and "shapiro_p" not in locals()
539         and "jb_p" in locals()
540         and jb_p > 0.05
541     ):
542         is_normal = True
543
544     if is_normal:
545         x = np.linspace(min(data), max(data), 1000)
546         y = stats.norm.pdf(x, mean, std)
547         bin_width = bins[1] - bins[0]
548         y_scaled = y * len(data) * bin_width
549
550         plt.plot(
551             x,
552             y_scaled,
553             "b-",
554             linewidth=2,
555             label=f"正态分布拟合 (={mean:.1f}, ={std:.1f})",
556         )
557
558         print(" 已在图中绘制正态分布拟合曲线")
559     else:
560         print(" 数据不符合正态分布，未绘制正态分布曲线")
561
562     plt.xlabel("天数", fontsize=14, fontweight="bold")
563     plt.ylabel("人数", fontsize=14, fontweight="bold")
564     plt.title(
565         f"BMI区间 {bmi_cat} 的孕妇Y染色体达标情况分布及正态分布检验",
566         fontsize=16,
567         fontweight="bold",
568     )
569     plt.legend(fontsize=12)

```



```

570     plt.grid(True, alpha=0.3)
571
572     plt.xticks(fontsize=12)
573     plt.yticks(fontsize=12)
574
575     plt.tight_layout()
576     normal_suffix = "_normal" if is_normal else "_non_normal"
577     plt.savefig(
578         f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_stacked_with_normal_test'
579         f'{normal_suffix}.png',
580         dpi=300,
581         bbox_inches="tight",
582     )
583     plt.close()
584
585     print(f"BMI 区间 {bmi_cat} 详细统计:")
586     print(f"    不能达标: {len(df_cannot)} 人")
587     print(f"    中间达标: {len(df_middle_cat)} 人")
588     print(f"    始终达标: {len(df_always)} 人")
589     print(f"    总计: {len(data)} 人")
590
591 def generate_stacked_bmi_mixed_charts():
592     df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
593     df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
594     df_always_can_test = pd.read_excel(
595         "./python_code/bmi_Y_always_can_test_result.xlsx"
596     )
597
598     df_middle["category"] = "middle"
599     df_cannot_test["category"] = "cannot"
600     df_always_can_test["category"] = "always_can"
601
602     df_all = pd.concat(
603         [df_middle, df_cannot_test, df_always_can_test], ignore_index=True
604     )
605
606     def categorize_bmi(bmi):
607         if bmi < 30.26:
608             return "<30.26"
609         elif 30.26 <= bmi < 32.30:
610             return "30.26-32.30"
611         elif 32.30 <= bmi < 34.92:
612             return "32.30-34.92"
613         elif 34.92 <= bmi < 39.49:
614             return "34.92-39.49"
615         else:
616             return ">39.49"
617
618     df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)
619
620     bmi_categories = ["<30.26", "30.26-32.30", "32.30-34.92", "34.92-39.49", ">39.49"]
621
622     categories = ["cannot", "middle", "always_can"]
623     colors = {"cannot": "lightcoral", "middle": "gold", "always_can": "seagreen"}

```

```

624 labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}
625
626 for bmi_cat in bmi_categories:
627     df_bmi = df_all[df_all["bmi_category"] == bmi_cat]
628
629     if df_bmi.empty:
630         print(f"警告: BMI区间 {bmi_cat} 没有数据")
631         continue
632
633     fig, ax1 = plt.subplots(figsize=(12, 8))
634
635     all_days = []
636
637     days_data = {}
638     for category in categories:
639         df_category = df_bmi[df_bmi["category"] == category]
640
641         if not df_category.empty:
642             if category == "cannot":
643                 days = df_category["最晚不达标天数"].tolist()
644             elif category == "middle":
645                 days = df_category["预测达标天数"].tolist()
646             else:
647                 days = df_category["最早达标天数"].tolist()
648
649             days_data[category] = days
650             all_days.extend(days)
651
652     stacked_data = []
653     stacked_labels = []
654     stacked_colors = []
655
656     if "cannot" in days_data:
657         stacked_data.append(days_data["cannot"])
658         stacked_labels.append(labels["cannot"])
659         stacked_colors.append(colors["cannot"])
660
661     if "middle" in days_data:
662         stacked_data.append(days_data["middle"])
663         stacked_labels.append(labels["middle"])
664         stacked_colors.append(colors["middle"])
665
666     if "always_can" in days_data:
667         stacked_data.append(days_data["always_can"])
668         stacked_labels.append(labels["always_can"])
669         stacked_colors.append(colors["always_can"])
670
671     if stacked_data:
672         if len(all_days) > 0:
673             data_range = max(all_days) - min(all_days)
674             bins_count = min(100, max(50, int(data_range / 3)))
675         else:
676             bins_count = 50
677
678     n, bins, patches = ax1.hist(

```

```

679         stacked_data,
680         bins=bins_count,
681         stacked=True,
682         color=stacked_colors,
683         label=stacked_labels,
684         alpha=0.95,
685         edgecolor="black",
686         linewidth=0.1,
687     )
688
689     ax1.set_xlabel("天数", fontsize=12)
690     ax1.set_ylabel("人数", fontsize=12)
691     ax1.set_title(
692         f"BMI 区间 {bmi_cat} 的孕妇Y染色体达标情况分布 (堆叠图)", fontsize=14
693     )
694     ax1.legend(fontsize=10)
695     ax1.grid(True, alpha=0.3)
696
697     if len(all_days) > 1:
698         ax2 = ax1.twinx()
699
700         try:
701             kde = stats.gaussian_kde(all_days)
702             x_range = np.linspace(min(all_days), max(all_days), 1000)
703             kde_values = kde(x_range)
704
705             ax2.plot(
706                 x_range,
707                 kde_values,
708                 color="dodgerblue",
709                 linewidth=2,
710                 linestyle="--",
711                 label="总体概率密度",
712             )
713
714             ax2.set_ylabel("概率密度", fontsize=12)
715             ax2.legend(bbox_to_anchor=(1.0, 0.9), fontsize=10)
716         except Exception as e:
717             print(f"BMI 区间 {bmi_cat} 计算概率密度时出错: {e}")
718
719     plt.tight_layout()
720
721     filename = f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt").replace("-", "_")}_stacked_distribution.png'
722     plt.savefig(filename, dpi=300, bbox_inches="tight")
723     plt.close()
724
725     print(f"BMI 区间 {bmi_cat} 统计:")
726     total_count = len(df_bmi)
727     for category in categories:
728         count = len(df_bmi[df_bmi["category"] == category])
729         percentage = (count / total_count) * 100 if total_count > 0 else 0
730         print(f" {labels[category]}: {count} 人 ({percentage:.1f}%)")
731     print(f" 总计: {total_count} 人")
732     print()

```

```

733
734
735 def generate_overall_stacked_bmi_chart():
736     df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
737     df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
738     df_always_can_test = pd.read_excel(
739         "./python_code/bmi_Y_always_can_test_result.xlsx"
740     )
741
742     df_middle["category"] = "middle"
743     df_cannot_test["category"] = "cannot"
744     df_always_can_test["category"] = "always_can"
745
746     df_all = pd.concat(
747         [df_middle, df_cannot_test, df_always_can_test], ignore_index=True
748     )
749
750     def categorize_bmi(bmi):
751         if bmi < 30.26:
752             return "<30.26"
753         elif 30.26 <= bmi < 32.30:
754             return "30.26-32.30"
755         elif 32.30 <= bmi < 34.92:
756             return "32.30-34.92"
757         elif 34.92 <= bmi < 39.49:
758             return "34.92-39.49"
759         else:
760             return ">39.49"
761
762     df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)
763
764     bmi_categories = ["<30.26", "30.26-32.30", "32.30-34.92", "34.92-39.49", ">39.49"]
765
766     categories = ["cannot", "middle", "always_can"]
767     colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
768     labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}
769
770     fig, ax1 = plt.subplots(figsize=(15, 8))
771
772     all_data_by_category = {cat: [] for cat in categories}
773
774     for bmi_cat in bmi_categories:
775         df_bmi = df_all[df_all["bmi_category"] == bmi_cat]
776
777         for category in categories:
778             df_category = df_bmi[df_bmi["category"] == category]
779
780             if not df_category.empty:
781                 if category == "cannot":
782                     days = df_category["最晚不达标天数"].tolist()
783                 elif category == "middle":
784                     days = df_category["预测达标天数"].tolist()
785                 else:
786                     days = df_category["最早达标天数"].tolist()
787

```

```

788         labeled_days = [(day, bmi_cat) for day in days]
789         all_data_by_category[category].extend(labeled_days)
790
791     x_positions = np.arange(len(bmi_categories))
792     bar_width = 0.6
793
794     bottom_values = np.zeros(len(bmi_categories))
795
796     for category in categories:
797         heights = []
798         for bmi_cat in bmi_categories:
799             count = len([x for x in all_data_by_category[category] if x[1] == bmi_cat])
800             heights.append(count)
801
802         ax1.bar(
803             x_positions,
804             heights,
805             bar_width,
806             bottom=bottom_values,
807             label=labels[category],
808             color=colors[category],
809             alpha=0.7,
810             edgecolor="black",
811             linewidth=0.5,
812         )
813
814         bottom_values = np.add(bottom_values, heights)
815
816     ax1.set_xlabel("BMI 区间", fontsize=12)
817     ax1.set_ylabel("人数", fontsize=12)
818     ax1.set_title("各BMI区间孕妇Y染色体达标情况分布（堆叠柱状图）", fontsize=14)
819     ax1.set_xticks(x_positions)
820     ax1.set_xticklabels(bmi_categories)
821     ax1.legend(fontsize=10)
822     ax1.grid(True, alpha=0.3, axis="y")
823
824     plt.tight_layout()
825
826     plt.savefig(
827         "./python_code/BMI_all_intervals_stacked_distribution_new.png",
828         dpi=300,
829         bbox_inches="tight",
830     )
831     plt.close()
832
833     print("总体堆叠柱状图已生成")
834
835
836 if __name__ == "__main__":
837     generate_overall_stacked_bmi_chart()
838     generate_stacked_bmi_mixed_charts()
839     print("\n图表生成完成!")

```

Listing 7: C(2) 聚类.py

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.cluster import KMeans
5 from sklearn.preprocessing import StandardScaler
6 from sklearn.metrics import (
7     silhouette_score,
8     davies_bouldin_score,
9     calinski_harabasz_score,
10 )
11
12 plt.rcParams["font.sans-serif"] = ["SimHei"]
13 plt.rcParams["axes.unicode_minus"] = False
14
15
16 def perform_clustering(df, n_clusters=5):
17     X = df[["BMI"]].values
18     scaler = StandardScaler()
19     X_scaled = scaler.fit_transform(X)
20
21     kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
22     clusters = kmeans.fit_predict(X_scaled)
23
24     df["Cluster"] = clusters
25
26     centers_scaled = kmeans.cluster_centers_
27     centers_original = scaler.inverse_transform(centers_scaled)
28
29     return df, kmeans, scaler, centers_original, X_scaled
30
31
32 def evaluate_clustering(X, clusters):
33     silhouette_avg = silhouette_score(X, clusters)
34     dbi = davies_bouldin_score(X, clusters)
35     ch_index = calinski_harabasz_score(X, clusters)
36
37     print("\n聚类效果评估:")
38     print(f"轮廓系数 (Silhouette Score): {silhouette_avg:.4f}")
39     print(f"DBI指数 (Davies-Bouldin Index): {dbi:.4f}")
40     print(f"CH指数 (Calinski-Harabasz Index): {ch_index:.4f}")
41
42     return silhouette_avg, dbi, ch_index
43
44
45 def print_cluster_ranges(df):
46     print("\n各聚类的BMI范围:")
47     print("-" * 50)
48     for i in range(5):
49         cluster_data = df[df["Cluster"] == i]["BMI"]
50         min_bmi = cluster_data.min()
51         max_bmi = cluster_data.max()
52         mean_bmi = cluster_data.mean()
53         count = len(cluster_data)
54
55         if mean_bmi < 18.5:

```

```

56         category = "偏瘦"
57     elif mean_bmi < 24:
58         category = "正常"
59     elif mean_bmi < 28:
60         category = "偏胖"
61     else:
62         category = "肥胖"
63
64     print(
65         f"聚类 {i}: {count:3d} 个样本, BMI范围: {min_bmi:.2f}--{max_bmi:.2f}, "
66         f"平均值: {mean_bmi:.2f} ({category})"
67     )
68
69     print("-" * 50)
70
71
72 def visualize_results(df, centers_original, silhouette_avg, dbi, ch_index):
73     fig, axes = plt.subplots(2, 2, figsize=(15, 12))
74
75     for i in range(5):
76         cluster_data = df[df["Cluster"] == i]["BMI"]
77         axes[0, 0].hist(cluster_data, alpha=0.6, bins=20, label=f"Cluster {i}")
78
79         axes[0, 0].axvline(x=18.5, color="r", linestyle="--", label="偏瘦阈值")
80         axes[0, 0].axvline(x=24, color="g", linestyle="--", label="正常阈值")
81         axes[0, 0].axvline(x=28, color="orange", linestyle="--", label="偏胖阈值")
82         axes[0, 0].set_xlabel("BMI")
83         axes[0, 0].set_ylabel("频数")
84         axes[0, 0].set_title("各聚类BMI分布")
85         axes[0, 0].legend()
86         axes[0, 0].grid(True, alpha=0.3)
87
88         cluster_labels = [f"Cluster {i}" for i in range(5)]
89         axes[0, 1].bar(
90             cluster_labels,
91             centers_original.flatten(),
92             color=["blue", "green", "red", "purple", "orange"],
93         )
94         axes[0, 1].set_ylabel("BMI值")
95         axes[0, 1].set_title("各聚类中心BMI值")
96         axes[0, 1].grid(True, alpha=0.3)
97
98         for i in range(5):
99             cluster_data = df[df["Cluster"] == i]
100             axes[1, 0].scatter(
101                 cluster_data["BMI"],
102                 cluster_data["对应达标天数"],
103                 alpha=0.6,
104                 label=f"Cluster {i}",
105             )
106
107         axes[1, 0].set_xlabel("BMI")
108         axes[1, 0].set_ylabel("达标天数")
109         axes[1, 0].set_title("BMI与达标天数的关系")
110         axes[1, 0].legend()

```

```

111     axes[1, 0].grid(True, alpha=0.3)
112
113
114 def save_results(df, centers_original, silhouette_avg, dbi, ch_index):
115     summary_data = {
116         "聚类编号": range(5),
117         "聚类中心(BMI)": centers_original.flatten(),
118         "样本数量": [len(df[df["Cluster"] == i]) for i in range(5)],
119         "BMI最小值": [df[df["Cluster"] == i]["BMI"].min() for i in range(5)],
120         "BMI最大值": [df[df["Cluster"] == i]["BMI"].max() for i in range(5)],
121         "BMI平均值": [df[df["Cluster"] == i]["BMI"].mean() for i in range(5)],
122     }
123     summary_df = pd.DataFrame(summary_data)
124
125     eval_df = pd.DataFrame(
126         {
127             "评估指标": ["轮廓系数", "DBI指数", "CH指数"],
128             "值": [silhouette_avg, dbi, ch_index],
129         }
130     )
131
132     with pd.ExcelWriter("聚类分析结果.xlsx") as writer:
133         df.to_excel(writer, sheet_name="原始数据与聚类结果", index=False)
134         summary_df.to_excel(writer, sheet_name="聚类摘要", index=False)
135         eval_df.to_excel(writer, sheet_name="评估指标", index=False)
136
137     print("\n结果已保存到 '聚类分析结果.xlsx'")
138
139
140 def main():
141     file_path = "python_code/bmi_Y_result.xlsx"
142     df = pd.read_excel(file_path)
143     df_result, kmeans, scaler, centers_original, X_scaled = perform_clustering(df)
144
145     silhouette_avg, dbi, ch_index = evaluate_clustering(
146         X_scaled, df_result["Cluster"].values
147     )
148
149     print_cluster_ranges(df_result)
150
151     visualize_results(df_result, centers_original, silhouette_avg, dbi, ch_index)
152
153     save_results(df_result, centers_original, silhouette_avg, dbi, ch_index)
154
155
156 if __name__ == "__main__":
157     main()

```