一

二

三

四

五

六

七

八

九

十

1

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

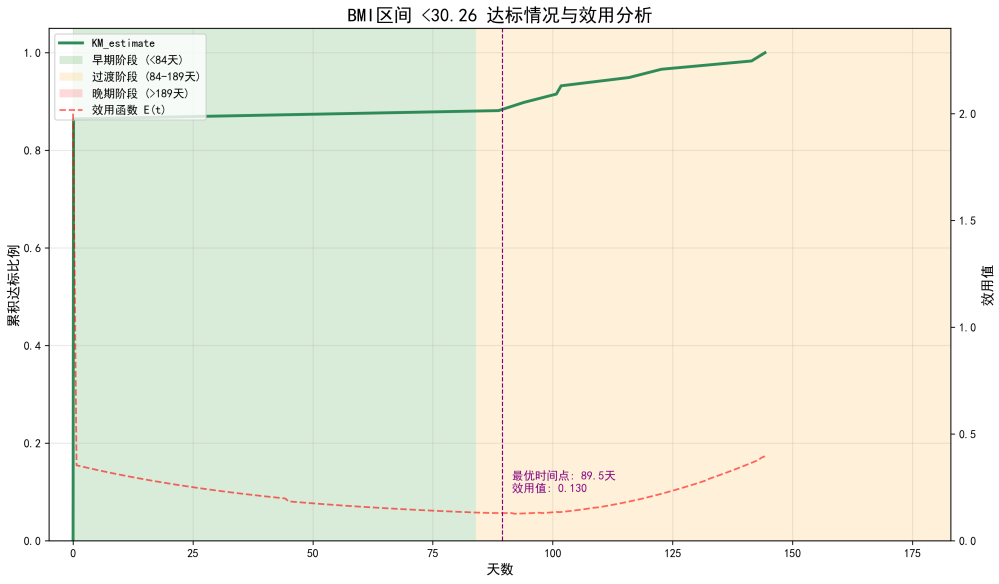

图 1: 累积达标曲线（小于 30.26 组）

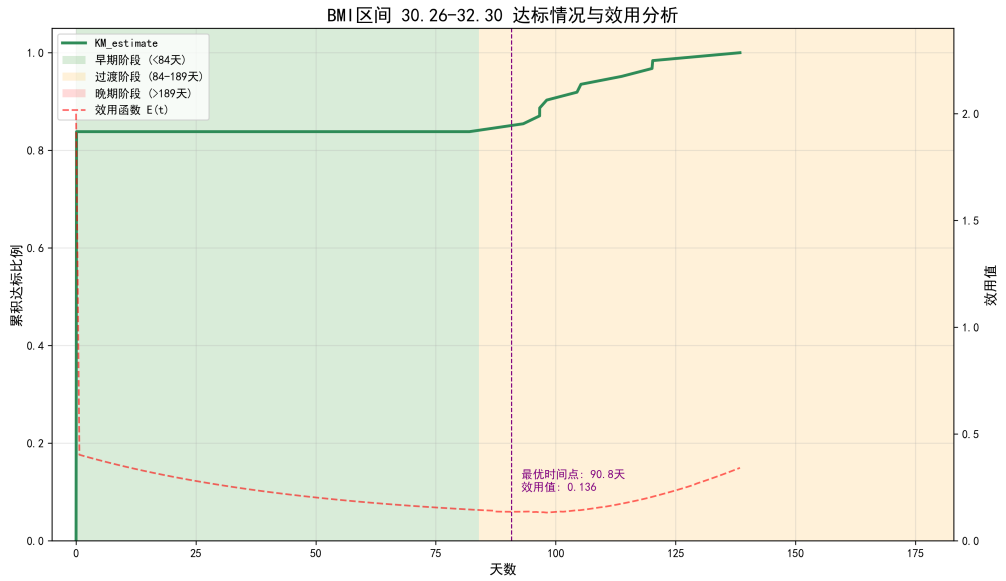

图 2: 累积达标曲线（30.26-32.30 组）

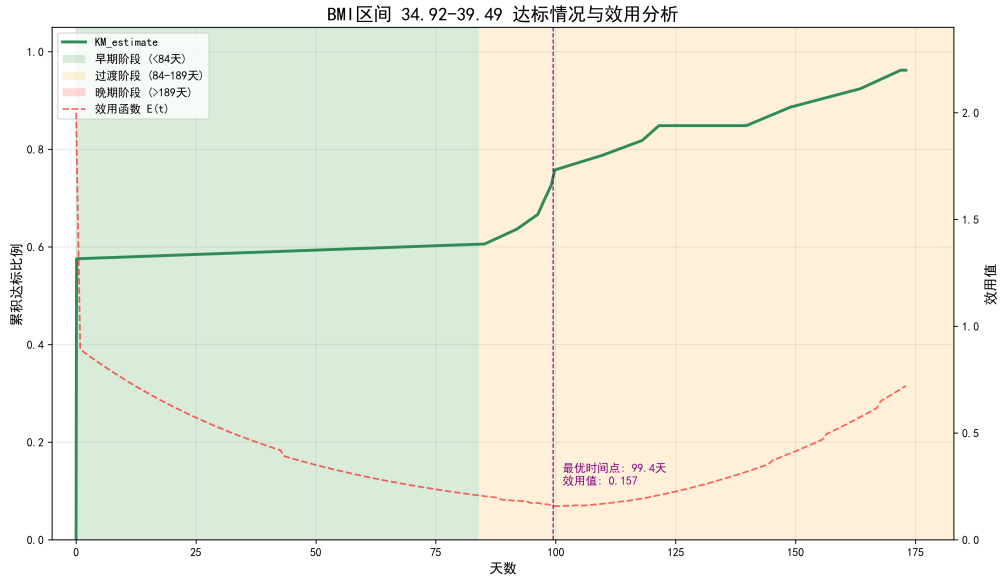

图 3: 累积达标曲线（34.92-39.49 组）

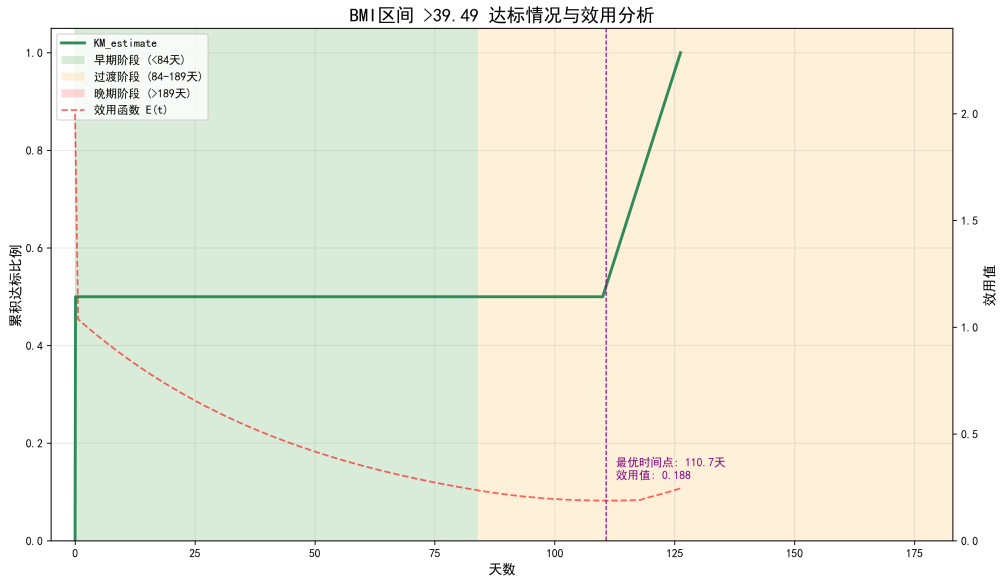

图 4: 累积达标曲线（大于 39.49 组）

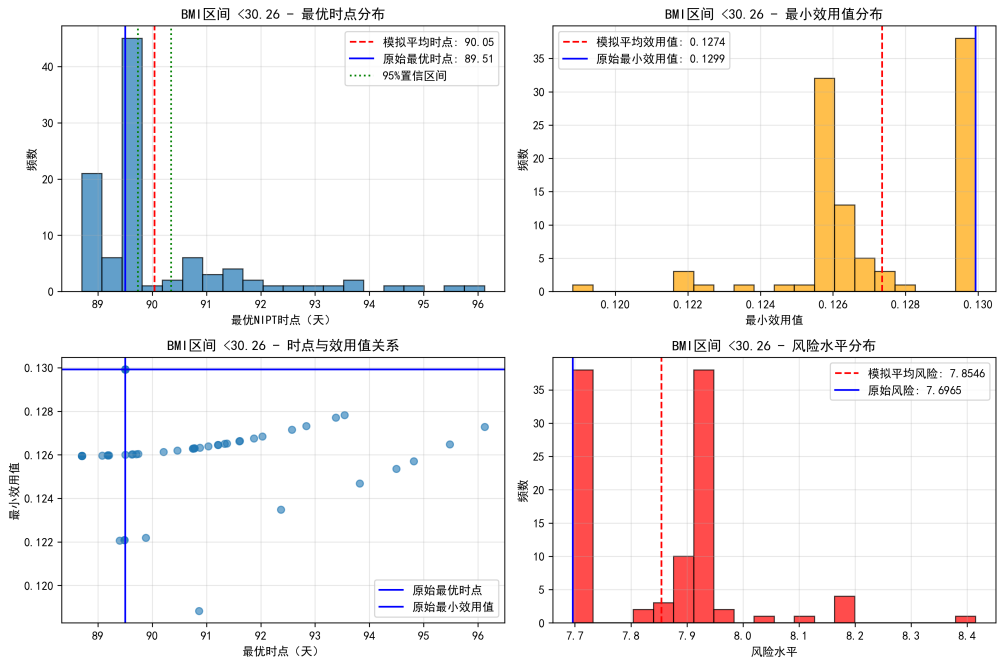

图 5: 误差模拟（小于 30.26 组）

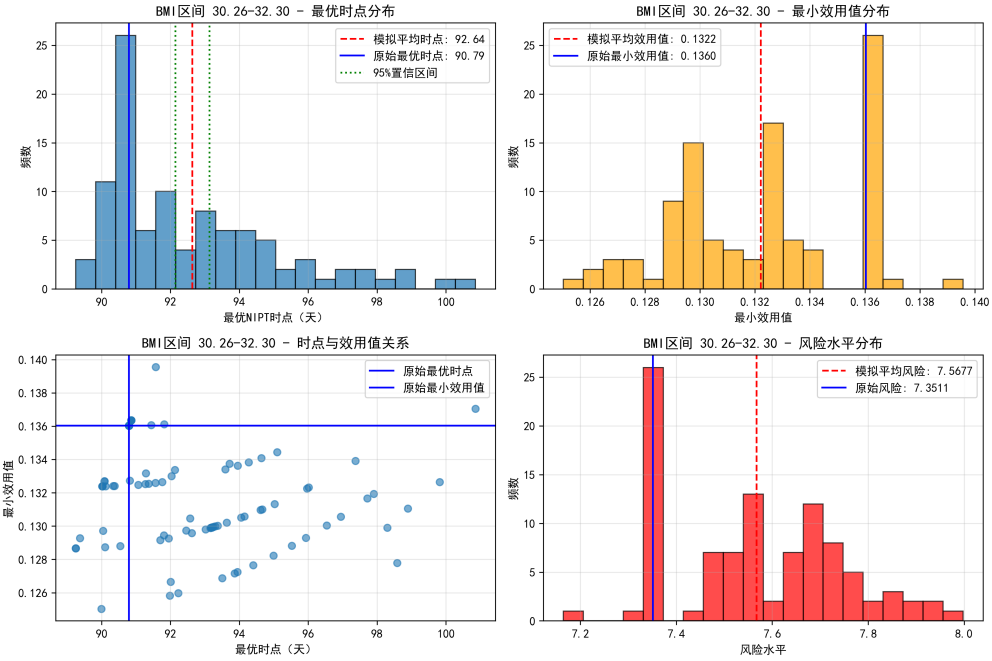

图 6: 误差模拟（30.26-32.30 组）

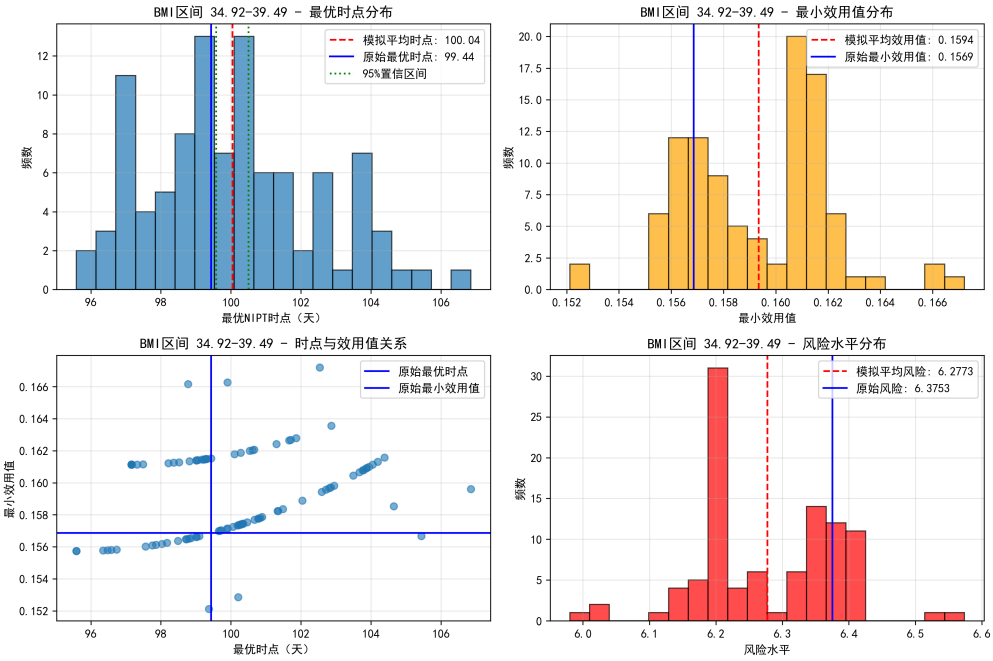

图 7: 误差模拟（34.92-39.49 组）

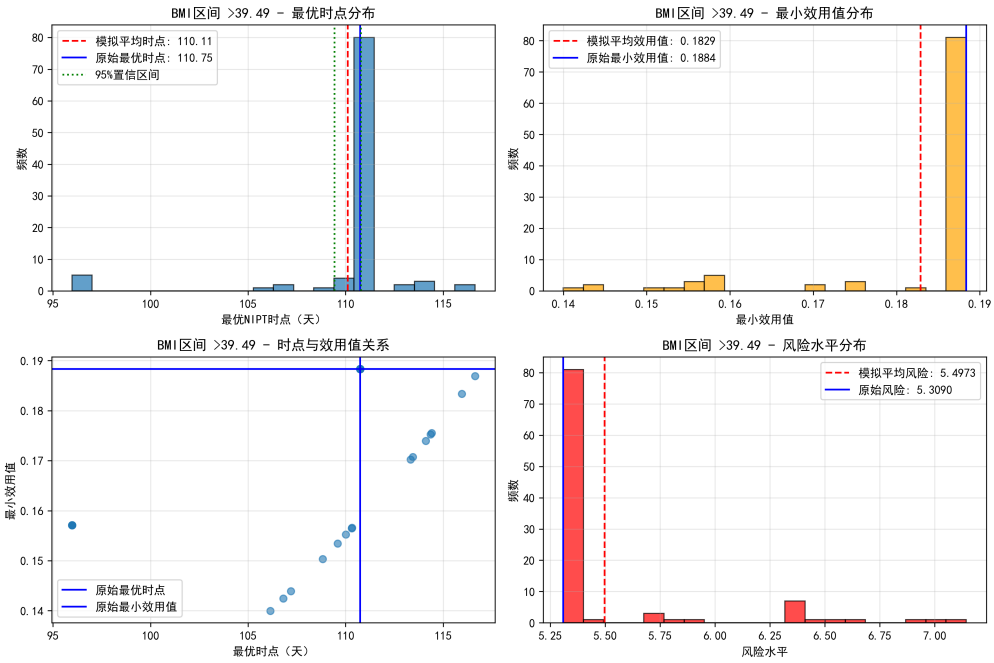

图 8: 误差模拟（大于 39.49 组）

Listing 1: 混合模型.py

```r
# 加载必要的包
library(readxl)
library(nlme)
library(ggplot2)
library(lme4)
library(MuMIn)
library(mgcv)  # 用于GAMM模型
library(merTools)
library(sjPlot)
library(gridExtra)
library(itsadug)  # 用于GAM模型可视化

# 读取数据
data_path <- "D:/math_modeling/25_C/mathmodeling_2024(1)/文件/副本C(1)_total.xlsx"
data <- read_excel(data_path)

# 查看数据结构
str(data)
head(data)

# 重命名列名以便使用
colnames(data) <- c("Subject", "BMI", "Y_concentration", "Gestational_days")

# 数据预处理
data$Subject <- as.factor(data$Subject)
data$BMI <- as.numeric(data$BMI)
data$Y_concentration <- as.numeric(data$Y_concentration)
data$Gestational_days <- as.numeric(data$Gestational_days)
```

```r

# 探索性数据分析
exploratory_plot <- ggplot(data, aes(x = Gestational_days, y = Y_concentration, color = Subject)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  theme_minimal() +
  labs(title = "Y染色体浓度随孕周变化", x = "孕周(天)", y = "Y染色体浓度")

print(exploratory_plot)
ggsave("exploratory_plot.png", width = 10, height = 6, dpi = 300)

# 1. 线性混合模型 (LMM)
cat("拟合线性混合模型...\n")
lmm_model <- lmer(Y_concentration ~ BMI + Gestational_days + (1|Subject), data = data)
cat("线性混合模型拟合完成\n")
print(summary(lmm_model))

# 2. 指数模型
cat("拟合指数模型...\n")
exp_model_formula <- function(bmi, days, a, b, c) {
  a * exp(b * bmi + c * days)
}

tryCatch({
  exp_nlme <- nlme(Y_concentration ~ exp_model_formula(BMI, Gestational_days, a, b, c),
                   data = data,
                   fixed = a + b + c ~ 1,
                   random = a ~ 1 | Subject,
                   start = list(a = 0.05, b = 0.001, c = 0.001),
                   control = nlmeControl(maxIter = 2000, pnlsTol = 1e-4, msTol = 1e-4))
  cat("指数模型拟合成功\n")
  print(summary(exp_nlme))
}, error = function(e) {
  message("指数模型拟合失败: ", e$message)
  exp_nlme <- NULL
})

# 3. Logistic生长模型
cat("拟合Logistic模型...\n")
logistic_model_formula <- function(bmi, days, a, b, c, d) {
  a / (1 + exp(-(b * bmi + c * days - d)))
}

tryCatch({
  logistic_nlme <- nlme(Y_concentration ~ logistic_model_formula(BMI, Gestational_days, a, b, c, d),
                        data = data,
                        fixed = a + b + c + d ~ 1,
                        random = a ~ 1 | Subject,
                        start = list(a = 0.1, b = 0.01, c = 0.01, d = 50),
                        control = nlmeControl(maxIter = 2000, pnlsTol = 1e-4, msTol = 1e-4))
  cat("Logistic模型拟合成功\n")
  print(summary(logistic_nlme))
}, error = function(e) {
  message("Logistic模型拟合失败: ", e$message)
  logistic_nlme <- NULL
```

```
})

# 4. Gompertz生长模型
cat("拟合Gompertz模型...\n")
gompertz_model_formula <- function(bmi, days, a, b, c, d) {
  a * exp(-exp(-(b * bmi + c * days - d)))
}

tryCatch({
  gompertz_nlme <- nlme(Y_concentration ~ gompertz_model_formula(BMI, Gestational_days, a, b, c, d),
                        data = data,
                        fixed = a + b + c + d ~ 1,
                        random = a ~ 1 | Subject,
                        start = list(a = 0.1, b = 0.01, c = 0.01, d = 50),
                        control = nlmeControl(maxIter = 2000, pnlsTol = 1e-4, msTol = 1e-4))
  cat("Gompertz模型拟合成功\n")
  print(summary(gompertz_nlme))
}, error = function(e) {
  message("Gompertz模型拟合失败: ", e$message)
  gompertz_nlme <- NULL
})

# 5. 广义加性混合模型 (GAMM)
cat("拟合广义加性混合模型...\n")
tryCatch({
  # 使用bam函数拟合GAMM, 适用于大型数据集
  gamm_model <- bam(Y_concentration ~ s(BMI) + s(Gestational_days) + s(Subject, bs = "re"),
                    data = data,
                    method = "REML")
  cat("GAMM模型拟合成功\n")
  print(summary(gamm_model))

  # 绘制平滑项图
  png("gamm_smooth_terms.png", width = 10, height = 6, units = "in", res = 300)
  par(mfrow = c(1, 2))
  plot(gamm_model, select = 1, main = "BMI平滑项")
  plot(gamm_model, select = 2, main = "孕周平滑项")
  dev.off()
}, error = function(e) {
  message("GAMM模型拟合失败: ", e$message)
  gamm_model <- NULL
})

# 模型比较
cat("开始模型比较...\n")
model_comparison <- data.frame(
  Model = "线性混合模型",
  AIC = AIC(lmm_model),
  BIC = BIC(lmm_model),
  LogLik = as.numeric(logLik(lmm_model))
)

# 添加非线性模型（如果成功拟合）
if(exists("exp_nlme") && !is.null(exp_nlme)) {
  model_comparison <- rbind(model_comparison,
```

```
                            data.frame(Model = "指数模型",
                                       AIC = AIC(exp_nlme),
                                       BIC = BIC(exp_nlme),
                                       LogLik = logLik(exp_nlme)))
}

if(exists("logistic_nlme") && !is.null(logistic_nlme)) {
  model_comparison <- rbind(model_comparison,
                            data.frame(Model = "Logistic模型",
                                       AIC = AIC(logistic_nlme),
                                       BIC = BIC(logistic_nlme),
                                       LogLik = logLik(logistic_nlme)))
}

if(exists("gompertz_nlme") && !is.null(gompertz_nlme)) {
  model_comparison <- rbind(model_comparison,
                            data.frame(Model = "Gompertz模型",
                                       AIC = AIC(gompertz_nlme),
                                       BIC = BIC(gompertz_nlme),
                                       LogLik = logLik(gompertz_nlme)))
}

if(exists("gamm_model") && !is.null(gamm_model)) {
  model_comparison <- rbind(model_comparison,
                            data.frame(Model = "GAMM模型",
                                       AIC = AIC(gamm_model),
                                       BIC = BIC(gamm_model),
                                       LogLik = logLik(gamm_model)))
}

# 显示模型比较结果
print("模型比较结果:")
print(model_comparison)

# 可视化拟合结果
# 创建预测函数
predict_nlme <- function(model, newdata) {
  if(inherits(model, "nlme")) {
    return(predict(model, newdata = newdata, level = 0))
  } else if(inherits(model, "lmerMod")) {
    return(predict(model, newdata = newdata, re.form = NA))
  } else if(inherits(model, "bam")) {
    return(predict(model, newdata = newdata, exclude = "s(Subject)"))
  } else {
    return(rep(NA, nrow(newdata)))
  }
}

# 生成预测数据 - 修复：使用平均BMI值
pred_data <- data.frame(
  BMI = rep(mean(data$BMI), 100),  # 使用平均BMI值
  Gestational_days = seq(min(data$Gestational_days), max(data$Gestational_days), length.out = 100),
  Subject = rep(levels(data$Subject)[1], 100)  # 使用第一个受试者
)

```

```r
# 为每个模型生成预测
pred_data$lmm <- predict_nlme(lmm_model, pred_data)

if(exists("exp_nlme") && !is.null(exp_nlme)) {
  pred_data$exp <- predict_nlme(exp_nlme, pred_data)
}

if(exists("logistic_nlme") && !is.null(logistic_nlme)) {
  pred_data$logistic <- predict_nlme(logistic_nlme, pred_data)
}

if(exists("gompertz_nlme") && !is.null(gompertz_nlme)) {
  pred_data$gompertz <- predict_nlme(gompertz_nlme, pred_data)
}

if(exists("gamm_model") && !is.null(gamm_model)) {
  pred_data$gamm <- predict_nlme(gamm_model, pred_data)
}

# 创建每个模型的拟合图
plots <- list()

# 线性混合模型
plots[[1]] <- ggplot() +
  geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha = 0.6)
      +
  geom_line(data = pred_data, aes(x = Gestational_days, y = lmm, color = "总体拟合"), size = 1) +
  theme_minimal() +
  labs(title = "线性混合模型拟合", x = "孕周(天)", y = "Y染色体浓度")

# 指数模型
if(exists("exp_nlme") && !is.null(exp_nlme)) {
  plots[[2]] <- ggplot() +
    geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha =
        0.6) +
    geom_line(data = pred_data, aes(x = Gestational_days, y = exp, color = "总体拟合"), size = 1) +
    theme_minimal() +
    labs(title = "指数模型拟合", x = "孕周(天)", y = "Y染色体浓度")
}

# Logistic模型
if(exists("logistic_nlme") && !is.null(logistic_nlme)) {
  plots[[3]] <- ggplot() +
    geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha =
        0.6) +
    geom_line(data = pred_data, aes(x = Gestational_days, y = logistic, color = "总体拟合"), size = 1) +
    theme_minimal() +
    labs(title = "Logistic模型拟合", x = "孕周(天)", y = "Y染色体浓度")
}

# Gompertz模型
if(exists("gompertz_nlme") && !is.null(gompertz_nlme)) {
  plots[[4]] <- ggplot() +
    geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha =
        0.6) +
```

```
    geom_line(data = pred_data, aes(x = Gestational_days, y = gompertz, color = "总体拟合"), size = 1) +
    theme_minimal() +
    labs(title = "Gompertz模型拟合", x = "孕周(天)", y = "Y染色体浓度")
}

# GAMM模型
if(exists("gamm_model") && !is.null(gamm_model)) {
  plots[[5]] <- ggplot() +
    geom_point(data = data, aes(x = Gestational_days, y = Y_concentration, color = Subject), alpha =
        0.6) +
    geom_line(data = pred_data, aes(x = Gestational_days, y = gamm, color = "总体拟合"), size = 1) +
    theme_minimal() +
    labs(title = "GAMM模型拟合", x = "孕周(天)", y = "Y染色体浓度")
}

# 保存所有图表
for(i in seq_along(plots)) {
  ggsave(paste0("model_fit_", i, ".png"), plots[[i]], width = 10, height = 6, dpi = 300)
}

# 创建模型比较图
model_comp_plot <- ggplot(model_comparison, aes(x = reorder(Model, AIC), y = AIC, fill = Model)) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  labs(title = "模型比较 - AIC值", x = "模型", y = "AIC值") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_fill_brewer(palette = "Set2")

ggsave("model_comparison_aic.png", model_comp_plot, width = 10, height = 6, dpi = 300)

# 残差分析
residual_plots <- list()

# 线性混合模型残差
residual_plots[[1]] <- ggplot(data, aes(x = fitted(lmm_model), y = resid(lmm_model))) +
  geom_point(alpha = 0.6) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  theme_minimal() +
  labs(title = "线性混合模型残差图", x = "拟合值", y = "残差")

# 指数模型残差
if(exists("exp_nlme") && !is.null(exp_nlme)) {
  residual_plots[[2]] <- ggplot(data, aes(x = fitted(exp_nlme), y = resid(exp_nlme))) +
    geom_point(alpha = 0.6) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    theme_minimal() +
    labs(title = "指数模型残差图", x = "拟合值", y = "残差")
}

# GAMM模型残差
if(exists("gamm_model") && !is.null(gamm_model)) {
  residual_plots[[3]] <- ggplot(data, aes(x = fitted(gamm_model), y = resid(gamm_model))) +
    geom_point(alpha = 0.6) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    theme_minimal() +
```

```r
    labs(title = "GAMM模型残差图", x = "拟合值", y = "残差")
}

# 保存残差图
for(i in seq_along(residual_plots)) {
  ggsave(paste0("residual_plot_", i, ".png"), residual_plots[[i]], width = 10, height = 6, dpi = 300)
}

# 输出模型摘要
sink("model_summaries.txt")
cat("线性混合模型摘要:\n")
print(summary(lmm_model))

if(exists("exp_nlme") && !is.null(exp_nlme)) {
  cat("\n指数模型摘要:\n")
  print(summary(exp_nlme))
}

if(exists("logistic_nlme") && !is.null(logistic_nlme)) {
  cat("\nLogistic模型摘要:\n")
  print(summary(logistic_nlme))
}

if(exists("gompertz_nlme") && !is.null(gompertz_nlme)) {
  cat("\nGompertz模型摘要:\n")
  print(summary(gompertz_nlme))
}

if(exists("gamm_model") && !is.null(gamm_model)) {
  cat("\nGAMM模型摘要:\n")
  print(summary(gamm_model))
}

cat("\n模型比较:\n")
print(model_comparison)
sink()

# 显示最佳模型
if(nrow(model_comparison) > 0) {
  best_model_idx <- which.min(model_comparison$AIC)
  cat("根据AIC准则，最佳模型是:", model_comparison$Model[best_model_idx], "\n")

  # 绘制最佳模型拟合图
  if(model_comparison$Model[best_model_idx] == "线性混合模型") {
    best_plot <- plots[[1]]
  } else if(model_comparison$Model[best_model_idx] == "指数模型") {
    best_plot <- plots[[2]]
  } else if(model_comparison$Model[best_model_idx] == "Logistic模型") {
    best_plot <- plots[[3]]
  } else if(model_comparison$Model[best_model_idx] == "Gompertz模型") {
    best_plot <- plots[[4]]
  } else if(model_comparison$Model[best_model_idx] == "GAMM模型") {
    best_plot <- plots[[5]]
  }

```

```
  ggsave("best_model_fit.png", best_plot, width = 10, height = 6, dpi = 300)
}

# 保存工作空间
save.image("mixed_model_analysis.RData")

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

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
plt.rcParams["axes.unicode_minus"] = False


def calculate_ahp_weights(matrix):

    n = matrix.shape[0]

    row_geom_mean = np.prod(matrix, axis=1) ** (1 / n)

    weights = row_geom_mean / np.sum(row_geom_mean)

    aw = np.dot(matrix, weights)
    lambda_max = np.mean(aw / weights)

    ci = (lambda_max - n) / (n - 1)

    ri_table = {
        1: 0,
        2: 0,
        3: 0.58,
        4: 0.90,
        5: 1.12,
        6: 1.24,
        7: 1.32,
        8: 1.41,
        9: 1.45,
        10: 1.49,
    }
    ri = ri_table.get(n, 1.51)

    cr = ci / ri if ri != 0 else 0

    return weights, cr


def generate_stacked_bmi_mixed_charts_with_ahp_weights():
    judgment_matrix = np.array([[1, 4, 6], [1 / 4, 1, 2], [1 / 6, 1 / 2, 1]])

    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)
```

```
    bmi_categories = ["<30.17", "30.17-32.25", "32.25-34.70", "34.70-37.11", ">37.11"]

    stacking_order = ["cannot", "middle", "always_can"]
    colors = {"cannot": "lightcoral", "middle": "goldenrod", "always_can": "seagreen"}
    labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}

    for bmi_cat in bmi_categories:
        df_bmi = df_all[df_all["bmi_category"] == bmi_cat]

        if df_bmi.empty:
            continue

        fig, ax1 = plt.subplots(figsize=(12, 8))

        all_days_raw = df_bmi["days_raw"].tolist()

        stacked_data, stacked_labels, stacked_colors = [], [], []
        for category in stacking_order:
            days = df_bmi[df_bmi["category"] == category]["days_raw"].tolist()
            if days:
                stacked_data.append(days)
                stacked_labels.append(labels[category])
                stacked_colors.append(colors[category])

        if stacked_data:
            data_range = max(all_days_raw) - min(all_days_raw) if all_days_raw else 0
            bins_count = (
                min(100, max(50, int(data_range / 3))) if data_range > 0 else 50
            )
            ax1.hist(
                stacked_data,
                bins=bins_count,
                stacked=True,
                color=stacked_colors,
                label=stacked_labels,
                alpha=0.95,
                edgecolor="black",
                linewidth=0.1,
            )

        ax1.set_xlabel("天数", fontsize=12)
        ax1.set_ylabel("人数", fontsize=12)
        ax1.set_title(f"BMI区间 {bmi_cat} 分布（AHP业务逻辑权重）", fontsize=14)
        ax1.legend(loc="upper left", fontsize=10)
        ax1.grid(True, alpha=0.3)

        if len(all_days_raw) > 1:
            ax2 = ax1.twinx()
            days_for_kde = df_bmi["days_raw"].to_numpy()
            weights_for_kde = df_bmi["weight"].to_numpy()

            try:
                weighted_kde = stats.gaussian_kde(days_for_kde, weights=weights_for_kde)
                x_range = np.linspace(min(all_days_raw), max(all_days_raw), 1000)
```

```
                kde_values = weighted_kde(x_range)

                ax2.plot(
                    x_range,
                    kde_values,
                    color="dodgerblue",
                    linewidth=2,
                    linestyle="-",
                    label="AHP加权概率密度",
                )
                ax2.set_ylabel("加权概率密度", fontsize=12)
                ax2.legend(loc="upper right", fontsize=10)

                cumulative_prob = np.cumsum(kde_values) * (x_range[1] - x_range[0])
                prob_levels = [0.85, 0.90, 0.95]
                for prob in prob_levels:
                    try:
                        index = np.where(cumulative_prob >= prob)[0][0]
                        day_value = x_range[index]
                        ax2.axvline(
                            x=day_value, color="purple", linestyle="--", linewidth=1
                        )
                        ax2.text(
                            day_value + 0.5,
                            max(kde_values) * (1 - prob),
                            f"{int(prob*100)}% -> {day_value:.1f}天",
                            color="purple",
                            rotation=90,
                        )
                    except IndexError:
                        pass

            except Exception as e:
                print(f"BMI区间 {bmi_cat} 计算加权概率密度时出错: {e}")

        plt.tight_layout()
        filename = f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt").replace("-", "_")}
            _AHP_weighted_dist.png'
        plt.savefig(filename, dpi=300, bbox_inches="tight")
        plt.close()


generate_stacked_bmi_mixed_charts_with_ahp_weights()
```

Listing 3: C(2)(生存函数).py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from lifelines import KaplanMeierFitter
from scipy.optimize import minimize_scalar

plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
plt.rcParams["axes.unicode_minus"] = False

```

```python

def analyze_with_dynamic_utility():
    def get_rearly(t):
        return 2.0 * np.exp(-t / 50)

    def get_rlate(t):
        if t < 84:
            return 0.1
        elif t <= 189:
            normalized_t = (t - 84) / (189 - 84)
            return 0.1 + 0.9 * (normalized_t**2)
        else:
            return 1.0

    def calculate_utility(t, survival_func):
        if t < 0:
            return float("inf")
        closest_time_idx = np.abs(survival_func.index - t).argmin()
        actual_time = survival_func.index[closest_time_idx]
        p_t = 1 - survival_func.loc[actual_time, "s_t"]
        return (1 - p_t) * get_rearly(t) + p_t * get_rlate(t)

    df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
    df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
    df_always_can_test = pd.read_excel(
        "./python_code/bmi_Y_always_can_test_result.xlsx"
    )

    df_all = pd.concat(
        [
            df_middle.assign(category="middle"),
            df_cannot_test.assign(category="cannot"),
            df_always_can_test.assign(category="always_can"),
        ],
        ignore_index=True,
    )

    df_all["duration"] = 0.0
    df_all["event_observed"] = 0

    df_all.loc[df_all["category"] == "cannot", "duration"] = df_all["最晚不达标天数"]
    df_all.loc[df_all["category"] == "cannot", "event_observed"] = 0

    df_all.loc[df_all["category"] == "middle", "duration"] = df_all["预测达标天数"]
    df_all.loc[df_all["category"] == "middle", "event_observed"] = 1

    df_all.loc[df_all["category"] == "always_can", "duration"] = 0.1
    df_all.loc[df_all["category"] == "always_can", "event_observed"] = 1

    def categorize_bmi(bmi):
        if bmi < 30.26:
            return "<30.26"
        elif 30.26 <= bmi < 32.30:
            return "30.26-32.30"
        elif 32.30 <= bmi < 34.92:
```

```
            return "32.30-34.92"
        elif 34.92 <= bmi < 39.49:
            return "34.92-39.49"
        else:
            return ">39.49"

    df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)

    bmi_categories = ["<30.26", "30.26-32.30", "32.30-34.92", "34.92-39.49", ">39.49"]

    results_table = []

    for bmi_cat in bmi_categories:
        df_bmi = df_all[df_all["bmi_category"] == bmi_cat].copy()

        if df_bmi.empty:
            results_table.append(
                {
                    "BMI分组": bmi_cat,
                    "最优时点(天)": "无数据",
                    "最小效用值": "无数据",
                    "风险水平": "无数据",
                    "样本量": 0,
                }
            )
            continue

        print(f"--- 正在分析 BMI 区间: {bmi_cat} ---")

        kmf = KaplanMeierFitter()
        kmf.fit(durations=df_bmi["duration"], event_observed=df_bmi["event_observed"])

        s_t = kmf.survival_function_.rename(columns={"KM_estimate": "s_t"})

        def objective(t):
            return calculate_utility(t, s_t)

        result = minimize_scalar(
            objective, bounds=(0, df_bmi["duration"].max()), method="bounded"
        )

        optimal_time = result.x
        optimal_utility = result.fun
        risk_level = 1 / optimal_utility if optimal_utility > 0 else float("inf")
        sample_size = len(df_bmi)

        print(f"\n最优预测时间点分析:")
        print(f"  - 最优时间点: {optimal_time:.1f} 天")
        print(f"  - 最小效用值: {optimal_utility:.4f}")
        print(f"  - 风险水平: {risk_level:.4f}")
        print(f"  - 样本量: {sample_size}")

        plt.figure(figsize=(14, 8))
        ax = plt.gca()

```

```
        (1 - kmf.survival_function_).plot(
            ax=ax,
            label=f"累积达标函数 F(t) ({bmi_cat})",
            color="seagreen",
            linewidth=2.5,
        )

        time_points = np.linspace(0, df_bmi["duration"].max(), 200)
        utilities = [calculate_utility(t, s_t) for t in time_points]
        ax_twin = ax.twinx()
        ax_twin.plot(
            time_points, utilities, "--", color="red", label="效用函数 E(t)", alpha=0.6
        )

        ax_twin.axvline(x=optimal_time, color="purple", linestyle="--", linewidth=1)
        ax_twin.text(
            optimal_time + 2,
            min(utilities) + 0.1,
            f"最优时间点: {optimal_time:.1f}天\n效用值: {optimal_utility:.3f}",
            color="purple",
            fontsize=10,
        )

        ax.axvspan(0, 84, facecolor="green", alpha=0.15, label="早期阶段 (<84天)")
        ax.axvspan(84, 189, facecolor="orange", alpha=0.15, label="过渡阶段 (84-189天)")
        ax.axvspan(
            189,
            df_all["duration"].max() + 10,
            facecolor="red",
            alpha=0.15,
            label="晚期阶段 (>189天)",
        )

        ax.set_title(f"BMI区间 {bmi_cat} 达标情况与效用分析", fontsize=16)
        ax.set_xlabel("天数", fontsize=12)
        ax.set_ylabel("累积达标比例", fontsize=12)
        ax_twin.set_ylabel("效用值", fontsize=12)

        lines1, labels1 = ax.get_legend_handles_labels()
        lines2, labels2 = ax_twin.get_legend_handles_labels()
        ax.legend(lines1 + lines2, labels1 + labels2, loc="upper left")

        ax.grid(True, alpha=0.3)
        ax.set_ylim(0, 1.05)
        ax_twin.set_ylim(0, max(utilities) * 1.2)
        ax.set_xlim(-5, df_all["duration"].max() + 10)

        filename = f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt").replace("-", "_")}
            _utility_analysis.png'
        plt.savefig(filename, dpi=300, bbox_inches="tight")
        plt.close()

        results_table.append(
            {
                "BMI分组": bmi_cat,
```

```
                "最优时点(天)": f"{optimal_time:.1f}",
                "最小效用值": f"{optimal_utility:.4f}",
                "风险水平": f"{risk_level:.4f}",
                "样本量": sample_size,
            }
        )

        print(f"\n图表已保存至: {filename}")
        print("\n" + "=" * 50 + "\n")

    print("\n\n=== 各BMI分组NIPT时点计算结果 ===")
    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")
    print("-" * 70)

    for result in results_table:
        print(
            f"{result['BMI分组']}\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']}"
        )

    results_df = pd.DataFrame(results_table)
    results_df.to_excel("./python_code/NIPT_optimal_times_results.xlsx", index=False)
    print(f"\n结果表格已保存至: ./python_code/NIPT_optimal_times_results.xlsx")

    return results_df


results = analyze_with_dynamic_utility()
```

Listing 4: C(2)(正态分布检验).py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
from matplotlib import rcParams
from scipy import stats
import re

plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
plt.rcParams["axes.unicode_minus"] = False


def categorize_bmi(bmi):
    if bmi < 30:
        return "<30"
    elif 30 <= bmi < 32:
        return "30-32"
    elif 32 <= bmi < 34:
        return "32-34"
    elif 34 <= bmi < 36:
        return "34-36"
    else:
        return ">36"

```

```python

def normality_test_and_plot_with_qq():
    df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
    df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
    df_always_can_test = pd.read_excel(
        "./python_code/bmi_Y_always_can_test_result.xlsx"
    )

    df_middle["bmi_category"] = df_middle["BMI"].apply(categorize_bmi)
    df_cannot_test["bmi_category"] = df_cannot_test["BMI"].apply(categorize_bmi)
    df_always_can_test["bmi_category"] = df_always_can_test["BMI"].apply(categorize_bmi)

    bmi_categories = ["<30", "30-32", "32-34", "34-36", ">36"]

    colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
    labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}

    for bmi_cat in bmi_categories:
        df_cannot = df_cannot_test[df_cannot_test["bmi_category"] == bmi_cat]
        df_middle_cat = df_middle[df_middle["bmi_category"] == bmi_cat]
        df_always = df_always_can_test[df_always_can_test["bmi_category"] == bmi_cat]

        days_cannot = (
            df_cannot["最晚不达标天数"].tolist() if not df_cannot.empty else []
        )
        days_middle = (
            df_middle_cat["预测达标天数"].tolist() if not df_middle_cat.empty else []
        )
        days_always = df_always["最早达标天数"].tolist() if not df_always.empty else []

        all_days = days_cannot + days_middle + days_always

        if not all_days:
            print(f"警告: BMI区间 {bmi_cat} 没有数据")
            continue

        data = np.array(all_days)

        print(f"\n=== BMI区间 {bmi_cat} 的正态分布检验 ===")
        print(f"样本数量: {len(data)}")

        if len(data) <= 5000:
            try:
                shapiro_stat, shapiro_p = stats.shapiro(data)
                print(
                    f"Shapiro-Wilk检验: 统计量={shapiro_stat:.6f}, p值={shapiro_p:.6f}"
                )
                if shapiro_p > 0.05:
                    print("  结论: 数据符合正态分布 (p > 0.05)")
                else:
                    print("  结论: 数据不符合正态分布 (p <= 0.05)")
            except Exception as e:
                print(f"Shapiro-Wilk检验失败: {e}")
        else:
            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
```

```
            and "shapiro_p" not in locals()
            and "jb_p" in locals()
            and jb_p > 0.05
        ):
            is_normal = True

        if is_normal:
            x = np.linspace(min(data), max(data), 1000)
            y = stats.norm.pdf(x, mean, std)
            bin_width = bins[1] - bins[0]
            y_scaled = y * len(data) * bin_width

            plt.plot(
                x,
                y_scaled,
                "b-",
                linewidth=2,
                label=f"正态分布拟合 ( ={mean:.1f},  ={std:.1f})",
            )

            print("  已在图中绘制正态分布拟合曲线")
        else:
            print("  数据不符合正态分布，未绘制正态分布曲线")

        plt.xlabel("天数", fontsize=14, fontweight="bold")
        plt.ylabel("人数", fontsize=14, fontweight="bold")
        plt.title(
            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",
            fontsize=16,
            fontweight="bold",
        )
        plt.legend(fontsize=12)
        plt.grid(True, alpha=0.3)

        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)

        plt.tight_layout()
        normal_suffix = "_normal" if is_normal else "_non_normal"
        plt.savefig(
            f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_stacked_with_normal_test
                {normal_suffix}.png',
            dpi=300,
            bbox_inches="tight",
        )
        plt.close()

        plt.figure(figsize=(10, 8))

        stats.probplot(data, dist="norm", plot=plt)
        plt.title(f"BMI区间 {bmi_cat} 的Q-Q图", fontsize=16, fontweight="bold")
        plt.xlabel("理论分位数", fontsize=14)
        plt.ylabel("样本分位数", fontsize=14)
        plt.grid(True, alpha=0.3)

```

```
        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)

        theoretical_quantiles, sample_quantiles = stats.probplot(data, dist="norm")
        slope, intercept, r_value, p_value, std_err = stats.linregress(
            theoretical_quantiles[0], theoretical_quantiles[1]
        )
        r_squared = r_value**2

        plt.text(
            0.05,
            0.95,
            f"R*R = {r_squared:.4f}",
            transform=plt.gca().transAxes,
            fontsize=12,
            verticalalignment="top",
            bbox=dict(boxstyle="round", facecolor="white", alpha=0.8),
        )

        plt.tight_layout()
        plt.savefig(
            f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_qq_plot.png',
            dpi=300,
            bbox_inches="tight",
        )
        plt.close()

        print(f"  已生成Q-Q图 (R² = {r_squared:.4f})")

        print(f"BMI区间 {bmi_cat} 详细统计:")
        print(f"  不能达标: {len(df_cannot)} 人")
        print(f"  中间达标: {len(df_middle_cat)} 人")
        print(f"  始终达标: {len(df_always)} 人")
        print(f"  总计: {len(data)} 人")


if __name__ == "__main__":
    normality_test_and_plot_with_qq()

    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

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from lifelines import KaplanMeierFitter
from scipy.optimize import minimize_scalar
from scipy.stats import norm
import re

plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
plt.rcParams["axes.unicode_minus"] = False


```

```python
def analyze_with_error_simulation(num_simulations=100, error_std=0.01):
    # 定义效用函数（与之前相同）
    def get_rearly(t):
        return 2.0 * np.exp(-t / 50)

    def get_rlate(t):
        if t < 84:
            return 0.1
        elif t <= 189:
            normalized_t = (t - 84) / (189 - 84)
            return 0.1 + 0.9 * (normalized_t**2)
        else:
            return 1.0

    def calculate_utility(t, survival_func):
        if t < 0:
            return float("inf")
        closest_time_idx = np.abs(survival_func.index - t).argmin()
        actual_time = survival_func.index[closest_time_idx]
        p_t = 1 - survival_func.loc[actual_time, "s_t"]
        return (1 - p_t) * get_rearly(t) + p_t * get_rlate(t)

    # 辅助函数：清理文件名中的非法字符
    def clean_filename(name):
        # 替换文件名字符串中的非法字符
        return re.sub(r'[<>:"/\\|?*]', "_", name)

    # 加载数据
    df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
    df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
    df_always_can_test = pd.read_excel(
        "./python_code/bmi_Y_always_can_test_result.xlsx"
    )

    df_all = pd.concat(
        [
            df_middle.assign(category="middle"),
            df_cannot_test.assign(category="cannot"),
            df_always_can_test.assign(category="always_can"),
        ],
        ignore_index=True,
    )

    df_all["duration"] = 0.0
    df_all["event_observed"] = 0

    df_all.loc[df_all["category"] == "cannot", "duration"] = df_all["最晚不达标天数"]
    df_all.loc[df_all["category"] == "cannot", "event_observed"] = 0

    df_all.loc[df_all["category"] == "middle", "duration"] = df_all["预测达标天数"]
    df_all.loc[df_all["category"] == "middle", "event_observed"] = 1

    df_all.loc[df_all["category"] == "always_can", "duration"] = 0.1
    df_all.loc[df_all["category"] == "always_can", "event_observed"] = 1

```

```python
    def categorize_bmi(bmi):
        if bmi < 30.26:
            return "<30.26"
        elif 30.26 <= bmi < 32.30:
            return "30.26-32.30"
        elif 32.30 <= bmi < 34.92:
            return "32.30-34.92"
        elif 34.92 <= bmi < 39.49:
            return "34.92-39.49"
        else:
            return ">39.49"

    df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)
    bmi_categories = ["<30.26", "30.26-32.30", "32.30-34.92", "34.92-39.49", ">39.49"]

    results = {
        bmi_cat: {"times": [], "utilities": [], "risks": []}
        for bmi_cat in bmi_categories
    }

    original_results = {}
    for bmi_cat in bmi_categories:
        df_bmi = df_all[df_all["bmi_category"] == bmi_cat].copy()
        if df_bmi.empty:
            continue

        kmf = KaplanMeierFitter()
        kmf.fit(durations=df_bmi["duration"], event_observed=df_bmi["event_observed"])
        s_t = kmf.survival_function_.rename(columns={"KM_estimate": "s_t"})

        def objective(t):
            return calculate_utility(t, s_t)

        result = minimize_scalar(
            objective, bounds=(0, df_bmi["duration"].max()), method="bounded"
        )

        original_time = result.x
        original_utility = result.fun
        original_risk = 1 / original_utility if original_utility > 0 else float("inf")

        original_results[bmi_cat] = {
            "time": original_time,
            "utility": original_utility,
            "risk": original_risk,
        }

        print(f"BMI区间 {bmi_cat} 原始结果:")
        print(f"  最优时点: {original_time:.2f} 天")
        print(f"  最小效用值: {original_utility:.4f}")
        print(f"  风险水平: {original_risk:.4f}")

    # 开始模拟
    for simulation in range(num_simulations):
        print(f"正在进行第 {simulation+1} 次模拟...")
```

```
        df_simulated = df_all.copy()
        np.random.seed(simulation)

        mask_middle = df_simulated["category"] == "middle"
        mask_always = df_simulated["category"] == "always_can"

        if sum(mask_middle) > 0:
            error_middle = np.random.normal(
                0,
                error_std * df_simulated.loc[mask_middle, "预测达标天数"].mean(),
                size=sum(mask_middle),
            )
            df_simulated.loc[mask_middle, "duration"] += error_middle

        if sum(mask_always) > 0:
            error_always = np.random.normal(
                0,
                error_std * df_simulated.loc[mask_always, "最早达标天数"].mean(),
                size=sum(mask_always),
            )
            df_simulated.loc[mask_always, "duration"] += error_always

        mask_cannot = df_simulated["category"] == "cannot"
        if sum(mask_cannot) > 0:
            error_cannot = np.random.normal(
                0,
                error_std * df_simulated.loc[mask_cannot, "最晚不达标天数"].mean(),
                size=sum(mask_cannot),
            )
            df_simulated.loc[mask_cannot, "duration"] += error_cannot

        df_simulated["duration"] = df_simulated["duration"].clip(lower=0.1)

        for bmi_cat in bmi_categories:
            df_bmi = df_simulated[df_simulated["bmi_category"] == bmi_cat].copy()
            if df_bmi.empty:
                continue

            kmf = KaplanMeierFitter()
            kmf.fit(
                durations=df_bmi["duration"], event_observed=df_bmi["event_observed"]
            )
            s_t = kmf.survival_function_.rename(columns={"KM_estimate": "s_t"})

            def objective(t):
                return calculate_utility(t, s_t)

            result = minimize_scalar(
                objective, bounds=(0, df_bmi["duration"].max()), method="bounded"
            )

            optimal_time = result.x
            optimal_utility = result.fun
            risk = 1 / optimal_utility if optimal_utility > 0 else float("inf")
```

```
178
179             results[bmi_cat]["times"].append(optimal_time)
180             results[bmi_cat]["utilities"].append(optimal_utility)
181             results[bmi_cat]["risks"].append(risk)
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})
                        天"
212             )
213             print(f"  原始最小效用值: {original_utility:.4f}")
214             print(f"  模拟最小效用值均值: {mean_utility:.4f}")
215             print(f"  效用值标准差: {std_utility:.4f}")
216             print(f"  原始风险水平: {original_risk:.4f}")
217             print(f"  模拟风险水平均值: {mean_risk:.4f}")
218             print(f"  风险水平标准差: {std_risk:.4f}")
219
220             plt.figure(figsize=(12, 8))
221
222             plt.subplot(2, 2, 1)
223             plt.hist(times, bins=20, alpha=0.7, edgecolor="black")
224             plt.axvline(
225                 mean_time,
226                 color="r",
227                 linestyle="--",
228                 label=f"模拟平均时点: {mean_time:.2f}",
229             )
230             plt.axvline(
231                 original_time,
```

```
                color="b",
                linestyle="-",
                label=f"原始最优时点: {original_time:.2f}",
            )
            plt.axvline(
                time_confidence_interval[0],
                color="g",
                linestyle=":",
                label="95%置信区间",
            )
            plt.axvline(time_confidence_interval[1], color="g", linestyle=":")
            plt.xlabel("最优NIPT时点（天）")
            plt.ylabel("频数")
            plt.title(f"BMI区间 {bmi_cat} - 最优时点分布")
            plt.legend()
            plt.grid(True, alpha=0.3)

            plt.subplot(2, 2, 2)
            plt.hist(utilities, bins=20, alpha=0.7, edgecolor="black", color="orange")
            plt.axvline(
                mean_utility,
                color="r",
                linestyle="--",
                label=f"模拟平均效用值: {mean_utility:.4f}",
            )
            plt.axvline(
                original_utility,
                color="b",
                linestyle="-",
                label=f"原始最小效用值: {original_utility:.4f}",
            )
            plt.xlabel("最小效用值")
            plt.ylabel("频数")
            plt.title(f"BMI区间 {bmi_cat} - 最小效用值分布")
            plt.legend()
            plt.grid(True, alpha=0.3)

            plt.subplot(2, 2, 3)
            plt.scatter(times, utilities, alpha=0.6)
            plt.axvline(original_time, color="b", linestyle="-", label="原始最优时点")
            plt.axhline(
                original_utility, color="b", linestyle="-", label="原始最小效用值"
            )
            plt.xlabel("最优时点（天）")
            plt.ylabel("最小效用值")
            plt.title(f"BMI区间 {bmi_cat} - 时点与效用值关系")
            plt.legend()
            plt.grid(True, alpha=0.3)

            plt.subplot(2, 2, 4)
            plt.hist(risks, bins=20, alpha=0.7, edgecolor="black", color="red")
            plt.axvline(
                mean_risk,
                color="r",
                linestyle="--",
```

```
                label=f"模拟平均风险: {mean_risk:.4f}",
            )
            plt.axvline(
                original_risk,
                color="b",
                linestyle="-",
                label=f"原始风险: {original_risk:.4f}",
            )
            plt.xlabel("风险水平")
            plt.ylabel("频数")
            plt.title(f"BMI区间 {bmi_cat} - 风险水平分布")
            plt.legend()
            plt.grid(True, alpha=0.3)

            plt.tight_layout()

            safe_bmi_cat = clean_filename(bmi_cat)
            filename = f"./python_code/error_analysis_BMI_{safe_bmi_cat}.png"
            plt.savefig(filename, dpi=300, bbox_inches="tight")
            plt.close()

            data_df = pd.DataFrame({"time": times, "utility": utilities, "risk": risks})
            csv_filename = f"./python_code/error_analysis_BMI_{safe_bmi_cat}.csv"
            data_df.to_csv(csv_filename, index=False)

            print(f"  图表已保存至: {filename}")
            print(f"  数据已保存至: {csv_filename}")
        else:
            print(f"BMI区间 {bmi_cat}: 无数据")


analyze_with_error_simulation(num_simulations=100, error_std=0.05)
```

Listing 6: C(2) 分类画图.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
from matplotlib import rcParams
from scipy import stats
import re


plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]
plt.rcParams["axes.unicode_minus"] = False


def best_bmi_Y():
    def weeks_to_days(weeks_str):
        match = re.match(r"(\d+)[wW](?:\+(\d+))?", str(weeks_str), re.IGNORECASE)
        if match:
            weeks = int(match.group(1))
            days = int(match.group(2)) if match.group(2) else 0
            return weeks * 7 + days
```

```
        else:
            print(f"无法解析孕周格式: {weeks_str}")
            return None

    df = pd.read_excel("./python_code/附件.xlsx", sheet_name=0)
    df = df[["孕妇代码", "检测孕周", "孕妇BMI", "Y染色体浓度"]]
    bmi_avg = df.groupby("孕妇代码")["孕妇BMI"].mean()
    df["孕妇平均BMI"] = df["孕妇代码"].map(bmi_avg)
    df["检测孕周_天数"] = df["检测孕周"].apply(weeks_to_days)

    # print(df)
    code_col = "孕妇代码"
    day_col = "检测孕周_天数"
    bmi_col = "孕妇平均BMI"
    y_col = "Y染色体浓度"
    # df_can_test_codes = df[df[y_col] >= 0.04][code_col].unique() # not left
    # df_cannot_test_codes = df[~df[code_col].isin(df_can_test_codes)][code_col].unique()  # left
    # df_codes_cannot_test_before_codes = df[df[y_col] < 0.04][code_col].unique() # not right
    # df_always_can_test_codes = df[~df[code_col].isin(df_codes_cannot_test_before_codes)][code_col].
        unique() # right
    # df_middle_codes = np.setdiff1d(df_can_test_codes, df_always_can_test_codes)
    df_codes = df[code_col].unique()

    result_middle = []
    result_cannot_test = []
    result_always_can_test = []
    for code in df_codes:
        data = df[df[code_col] == code].sort_values(by=day_col)
        mean_bmi = data[bmi_col].mean()
        first_ok_idx = None
        for idx, row in data.iterrows():
            if row[y_col] >= 0.04:
                first_ok_idx = idx
                break

        if first_ok_idx is None:
            result_cannot_test.append(
                {code_col: code, "BMI": mean_bmi, "最晚不达标天数": data[day_col].max()}
            )
            continue
        else:
            later_data = data.loc[first_ok_idx:]
            if (later_data[y_col] < 0.04).any():
                continue
            else:
                if first_ok_idx == data.index[0]:
                    result_always_can_test.append(
                        {
                            code_col: code,
                            "BMI": mean_bmi,
                            "最早达标天数": data[day_col].min(),
                        }
                    )
                else:
                    ok_row = data.loc[first_ok_idx]
```

```
                    prev_row = data.loc[
                        data.index[data.index.get_loc(first_ok_idx) - 1]
                    ]

                    w1 = abs(ok_row[y_col] - 0.04)
                    w2 = abs(prev_row[y_col] - 0.04)
                    predicted_day = (w2 * ok_row[day_col] + w1 * prev_row[day_col]) / (
                        w1 + w2
                    )
                    result_middle.append(
                        {code_col: code, "BMI": mean_bmi, "预测达标天数": predicted_day}
                    )

    df_middle = pd.DataFrame(result_middle)
    df_cannot_test = pd.DataFrame(result_cannot_test)
    df_always_can_test = pd.DataFrame(result_always_can_test)

    df_middle.to_excel("./python_code/bmi_Y_middle_result.xlsx", index=False)
    df_cannot_test.to_excel("./python_code/bmi_Y_cannot_test_result.xlsx", index=False)
    df_always_can_test.to_excel(
        "./python_code/bmi_Y_always_can_test_result.xlsx", index=False
    )


def generate_bmi_category_charts():
    df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
    df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
    df_always_can_test = pd.read_excel(
        "./python_code/bmi_Y_always_can_test_result.xlsx"
    )

    df_middle["category"] = "middle"
    df_cannot_test["category"] = "cannot"
    df_always_can_test["category"] = "always_can"

    df_all = pd.concat(
        [df_middle, df_cannot_test, df_always_can_test], ignore_index=True
    )

    def categorize_bmi(bmi):
        if bmi < 30:
            return "<30"
        elif 30 <= bmi < 32:
            return "30-32"
        elif 32 <= bmi < 34:
            return "32-34"
        elif 34 <= bmi < 36:
            return "34-36"
        else:
            return ">36"

    df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)

    bmi_categories = ["<30", "30-32", "32-34", "34-36", ">36"]

```

```
    categories = ["cannot", "middle", "always_can"]
    colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
    labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}

    for bmi_cat in bmi_categories:
        df_bmi = df_all[df_all["bmi_category"] == bmi_cat]

        if df_bmi.empty:
            print(f"警告: BMI区间 {bmi_cat} 没有数据")
            continue

        plt.figure(figsize=(12, 8))

        for category in categories:
            df_category = df_bmi[df_bmi["category"] == category]

            if not df_category.empty:
                if category == "cannot":
                    days = df_category["最晚不达标天数"]
                elif category == "middle":
                    days = df_category["预测达标天数"]
                else:
                    days = df_category["最早达标天数"]

                plt.hist(
                    days,
                    bins=20,
                    alpha=0.7,
                    color=colors[category],
                    label=labels[category],
                    edgecolor="black",
                    linewidth=0.5,
                )

        plt.xlabel("天数", fontsize=12)
        plt.ylabel("人数", fontsize=12)
        plt.title(f"BMI区间 {bmi_cat} 的孕妇Y染色体达标情况分布", fontsize=14)
        plt.legend(fontsize=10)
        plt.grid(True, alpha=0.3)

        plt.savefig(
            f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_distribution.png',
            dpi=300,
            bbox_inches="tight",
        )
        plt.close()

        print(f"BMI区间 {bmi_cat} 统计:")
        for category in categories:
            count = len(df_bmi[df_bmi["category"] == category])
            print(f"  {labels[category]}: {count} 人")
        print()


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:
```

```
                    days = df_category["最早达标天数"]

                ax1.hist(
                    days,
                    bins=20,
                    alpha=0.7,
                    color=colors[category],
                    label=labels[category],
                    edgecolor="black",
                    linewidth=0.5,
                )

                all_days.extend(days.tolist())

        ax1.set_xlabel("天数", fontsize=12)
        ax1.set_ylabel("人数", fontsize=12)
        ax1.set_title(f"BMI区间 {bmi_cat} 的孕妇Y染色体达标情况分布", fontsize=14)
        ax1.legend(fontsize=10)
        ax1.grid(True, alpha=0.3)

        if len(all_days) > 1:
            ax2 = ax1.twinx()

            try:
                kde = stats.gaussian_kde(all_days)
                x_range = np.linspace(min(all_days), max(all_days), 1000)
                kde_values = kde(x_range)

                ax2.plot(
                    x_range,
                    kde_values,
                    color="blue",
                    linewidth=2,
                    linestyle="-",
                    label="总体概率密度",
                )

                ax2.set_ylabel("概率密度", fontsize=12)
                ax2.legend(loc="upper right", fontsize=10)
            except Exception as e:
                print(f"BMI区间 {bmi_cat} 计算概率密度时出错: {e}")

        plt.tight_layout()

        plt.savefig(
            f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_distribution.png',
            dpi=300,
            bbox_inches="tight",
        )
        plt.close()

        print(f"BMI区间 {bmi_cat} 统计:")
        total_count = len(df_bmi)
        for category in categories:
            count = len(df_bmi[df_bmi["category"] == category])
```

```
            percentage = (count / total_count) * 100 if total_count > 0 else 0
            print(f"  {labels[category]}: {count} 人 ({percentage:.1f}%)")
        print(f"  总计: {total_count} 人")
        print()


def generate_stacked_bmi_charts_alternative():
    df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
    df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
    df_always_can_test = pd.read_excel(
        "./python_code/bmi_Y_always_can_test_result.xlsx"
    )

    def categorize_bmi(bmi):
        if bmi < 30:
            return "<30"
        elif 30 <= bmi < 32:
            return "30-32"
        elif 32 <= bmi < 34:
            return "32-34"
        elif 34 <= bmi < 36:
            return "34-36"
        else:
            return ">36"

    df_middle["bmi_category"] = df_middle["BMI"].apply(categorize_bmi)
    df_cannot_test["bmi_category"] = df_cannot_test["BMI"].apply(categorize_bmi)
    df_always_can_test["bmi_category"] = df_always_can_test["BMI"].apply(categorize_bmi)

    bmi_categories = ["<30", "30-32", "32-34", "34-36", ">36"]

    colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
    labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}
    category_order = ["cannot", "middle", "always_can"]

    for bmi_cat in bmi_categories:
        df_cannot = df_cannot_test[df_cannot_test["bmi_category"] == bmi_cat]
        df_middle_cat = df_middle[df_middle["bmi_category"] == bmi_cat]
        df_always = df_always_can_test[df_always_can_test["bmi_category"] == bmi_cat]

        if df_cannot.empty and df_middle_cat.empty and df_always.empty:
            print(f"警告: BMI区间 {bmi_cat} 没有数据")
            continue

        plt.figure(figsize=(12, 8))

        days_cannot = (
            df_cannot["最晚不达标天数"].tolist() if not df_cannot.empty else []
        )
        days_middle = (
            df_middle_cat["预测达标天数"].tolist() if not df_middle_cat.empty else []
        )
        days_always = df_always["最早达标天数"].tolist() if not df_always.empty else []

        all_days = days_cannot + days_middle + days_always
```

```
        if not all_days:
            print(f"警告: BMI区间 {bmi_cat} 没有有效数据")
            continue

        bins = np.linspace(min(all_days), max(all_days), 21)
        data_to_plot = []
        plot_labels = []
        plot_colors = []

        if days_cannot:
            data_to_plot.append(days_cannot)
            plot_labels.append(labels["cannot"])
            plot_colors.append(colors["cannot"])

        if days_middle:
            data_to_plot.append(days_middle)
            plot_labels.append(labels["middle"])
            plot_colors.append(colors["middle"])

        if days_always:
            data_to_plot.append(days_always)
            plot_labels.append(labels["always_can"])
            plot_colors.append(colors["always_can"])

        if data_to_plot:
            plt.hist(
                data_to_plot,
                bins=bins,
                stacked=True,
                color=plot_colors,
                label=plot_labels,
                alpha=0.7,
                edgecolor="black",
                linewidth=0.3,
            )

        plt.xlabel("天数", fontsize=14, fontweight="bold")
        plt.ylabel("人数", fontsize=14, fontweight="bold")
        plt.title(
            f"BMI区间 {bmi_cat} 的孕妇Y染色体达标情况分布",
            fontsize=16,
            fontweight="bold",
        )
        plt.legend(fontsize=12)
        plt.grid(True, alpha=0.3)

        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)

        plt.tight_layout()
        plt.savefig(
            f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_stacked_v2.png',
            dpi=300,
            bbox_inches="tight",
        )
```

```
        plt.close()

        print(f"BMI区间 {bmi_cat} 统计:")
        print(f"  不能达标: {len(df_cannot)} 人")
        print(f"  中间达标: {len(df_middle_cat)} 人")
        print(f"  始终达标: {len(df_always)} 人")
        print(f"  总计: {len(df_cannot) + len(df_middle_cat) + len(df_always)} 人")
        print()


def categorize_bmi(bmi):
    """根据BMI值分类"""
    if bmi < 30:
        return "<30"
    elif 30 <= bmi < 32:
        return "30-32"
    elif 32 <= bmi < 34:
        return "32-34"
    elif 34 <= bmi < 36:
        return "34-36"
    else:
        return ">36"


def normality_test_and_plot():
    df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
    df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
    df_always_can_test = pd.read_excel(
        "./python_code/bmi_Y_always_can_test_result.xlsx"
    )

    df_middle["bmi_category"] = df_middle["BMI"].apply(categorize_bmi)
    df_cannot_test["bmi_category"] = df_cannot_test["BMI"].apply(categorize_bmi)
    df_always_can_test["bmi_category"] = df_always_can_test["BMI"].apply(categorize_bmi)

    bmi_categories = ["<30", "30-32", "32-34", "34-36", ">36"]

    colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
    labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}

    for bmi_cat in bmi_categories:
        df_cannot = df_cannot_test[df_cannot_test["bmi_category"] == bmi_cat]
        df_middle_cat = df_middle[df_middle["bmi_category"] == bmi_cat]
        df_always = df_always_can_test[df_always_can_test["bmi_category"] == bmi_cat]

        days_cannot = (
            df_cannot["最晚不达标天数"].tolist() if not df_cannot.empty else []
        )
        days_middle = (
            df_middle_cat["预测达标天数"].tolist() if not df_middle_cat.empty else []
        )
        days_always = df_always["最早达标天数"].tolist() if not df_always.empty else []

        all_days = days_cannot + days_middle + days_always

```

```
        if not all_days:
            print(f"警告：BMI区间 {bmi_cat} 没有数据")
            continue

        data = np.array(all_days)

        print(f"\n=== BMI区间 {bmi_cat} 的正态分布检验 ===")
        print(f"样本数量：{len(data)}")

        if len(data) <= 5000:
            try:
                shapiro_stat, shapiro_p = stats.shapiro(data)
                print(
                    f"Shapiro-Wilk检验：统计量={shapiro_stat:.6f}, p值={shapiro_p:.6f}"
                )
                if shapiro_p > 0.05:
                    print("  结论：数据符合正态分布 (p > 0.05)")
                else:
                    print("  结论：数据不符合正态分布 (p <= 0.05)")
            except Exception as e:
                print(f"Shapiro-Wilk检验失败：{e}")
        else:
            print("Shapiro-Wilk检验：样本量过大(>5000)，跳过该检验")

        try:
            jb_stat, jb_p = stats.jarque_bera(data)
            print(f"Jarque-Bera检验：统计量={jb_stat:.6f}, p值={jb_p:.6f}")
            if jb_p > 0.05:
                print("  结论：数据符合正态分布 (p > 0.05)")
            else:
                print("  结论：数据不符合正态分布 (p <= 0.05)")
        except Exception as e:
            print(f"Jarque-Bera检验失败：{e}")

        mean = np.mean(data)
        std = np.std(data)
        print(f"数据均值：{mean:.2f}, 标准差：{std:.2f}")

        plt.figure(figsize=(12, 8))

        data_to_plot = []
        plot_labels = []
        plot_colors = []

        if days_cannot:
            data_to_plot.append(days_cannot)
            plot_labels.append(labels["cannot"])
            plot_colors.append(colors["cannot"])

        if days_middle:
            data_to_plot.append(days_middle)
            plot_labels.append(labels["middle"])
            plot_colors.append(colors["middle"])

        if days_always:
```

```
            data_to_plot.append(days_always)
            plot_labels.append(labels["always_can"])
            plot_colors.append(colors["always_can"])

        if data_to_plot:
            n, bins, patches = plt.hist(
                data_to_plot,
                bins=20,
                stacked=True,
                color=plot_colors,
                label=plot_labels,
                alpha=0.7,
                edgecolor="black",
                linewidth=0.3,
            )

        is_normal = False
        if len(data) <= 5000 and "shapiro_p" in locals() and shapiro_p > 0.05:
            is_normal = True
        elif len(data) > 5000 and "jb_p" in locals() and jb_p > 0.05:
            is_normal = True
        elif (
            len(data) <= 5000
            and "shapiro_p" not in locals()
            and "jb_p" in locals()
            and jb_p > 0.05
        ):
            is_normal = True

        if is_normal:
            x = np.linspace(min(data), max(data), 1000)
            y = stats.norm.pdf(x, mean, std)
            bin_width = bins[1] - bins[0]
            y_scaled = y * len(data) * bin_width

            plt.plot(
                x,
                y_scaled,
                "b-",
                linewidth=2,
                label=f"正态分布拟合 ( ={mean:.1f},  ={std:.1f})",
            )

            print("  已在图中绘制正态分布拟合曲线")
        else:
            print("  数据不符合正态分布，未绘制正态分布曲线")

        plt.xlabel("天数", fontsize=14, fontweight="bold")
        plt.ylabel("人数", fontsize=14, fontweight="bold")
        plt.title(
            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",
            fontsize=16,
            fontweight="bold",
        )
        plt.legend(fontsize=12)
```

```
        plt.grid(True, alpha=0.3)

        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)

        plt.tight_layout()
        normal_suffix = "_normal" if is_normal else "_non_normal"
        plt.savefig(
            f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt")}_stacked_with_normal_test
                {normal_suffix}.png',
            dpi=300,
            bbox_inches="tight",
        )
        plt.close()

        print(f"BMI区间 {bmi_cat} 详细统计:")
        print(f"  不能达标: {len(df_cannot)} 人")
        print(f"  中间达标: {len(df_middle_cat)} 人")
        print(f"  始终达标: {len(df_always)} 人")
        print(f"  总计: {len(data)} 人")


def generate_stacked_bmi_mixed_charts():
    df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
    df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
    df_always_can_test = pd.read_excel(
        "./python_code/bmi_Y_always_can_test_result.xlsx"
    )

    df_middle["category"] = "middle"
    df_cannot_test["category"] = "cannot"
    df_always_can_test["category"] = "always_can"

    df_all = pd.concat(
        [df_middle, df_cannot_test, df_always_can_test], ignore_index=True
    )

    def categorize_bmi(bmi):
        if bmi < 30.26:
            return "<30.26"
        elif 30.26 <= bmi < 32.30:
            return "30.26-32.30"
        elif 32.30 <= bmi < 34.92:
            return "32.30-34.92"
        elif 34.92 <= bmi < 39.49:
            return "34.92-39.49"
        else:
            return ">39.49"

    df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)

    bmi_categories = ["<30.26", "30.26-32.30", "32.30-34.92", "34.92-39.49", ">39.49"]

    categories = ["cannot", "middle", "always_can"]
    colors = {"cannot": "lightcoral", "middle": "gold", "always_can": "seagreen"}
```

```
    labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}

    for bmi_cat in bmi_categories:
        df_bmi = df_all[df_all["bmi_category"] == bmi_cat]

        if df_bmi.empty:
            print(f"警告: BMI区间 {bmi_cat} 没有数据")
            continue

        fig, ax1 = plt.subplots(figsize=(12, 8))

        all_days = []

        days_data = {}
        for category in categories:
            df_category = df_bmi[df_bmi["category"] == category]

            if not df_category.empty:
                if category == "cannot":
                    days = df_category["最晚不达标天数"].tolist()
                elif category == "middle":
                    days = df_category["预测达标天数"].tolist()
                else:
                    days = df_category["最早达标天数"].tolist()

                days_data[category] = days
                all_days.extend(days)

        stacked_data = []
        stacked_labels = []
        stacked_colors = []

        if "cannot" in days_data:
            stacked_data.append(days_data["cannot"])
            stacked_labels.append(labels["cannot"])
            stacked_colors.append(colors["cannot"])

        if "middle" in days_data:
            stacked_data.append(days_data["middle"])
            stacked_labels.append(labels["middle"])
            stacked_colors.append(colors["middle"])

        if "always_can" in days_data:
            stacked_data.append(days_data["always_can"])
            stacked_labels.append(labels["always_can"])
            stacked_colors.append(colors["always_can"])

        if stacked_data:
            if len(all_days) > 0:
                data_range = max(all_days) - min(all_days)
                bins_count = min(100, max(50, int(data_range / 3)))
            else:
                bins_count = 50

            n, bins, patches = ax1.hist(
```

```
                stacked_data,
                bins=bins_count,
                stacked=True,
                color=stacked_colors,
                label=stacked_labels,
                alpha=0.95,
                edgecolor="black",
                linewidth=0.1,
            )

        ax1.set_xlabel("天数", fontsize=12)
        ax1.set_ylabel("人数", fontsize=12)
        ax1.set_title(
            f"BMI区间 {bmi_cat} 的孕妇Y染色体达标情况分布（堆叠图）", fontsize=14
        )
        ax1.legend(fontsize=10)
        ax1.grid(True, alpha=0.3)

        if len(all_days) > 1:
            ax2 = ax1.twinx()

            try:
                kde = stats.gaussian_kde(all_days)
                x_range = np.linspace(min(all_days), max(all_days), 1000)
                kde_values = kde(x_range)

                ax2.plot(
                    x_range,
                    kde_values,
                    color="dodgerblue",
                    linewidth=2,
                    linestyle="-",
                    label="总体概率密度",
                )

                ax2.set_ylabel("概率密度", fontsize=12)
                ax2.legend(bbox_to_anchor=(1.0, 0.9), fontsize=10)
            except Exception as e:
                print(f"BMI区间 {bmi_cat} 计算概率密度时出错: {e}")

        plt.tight_layout()

        filename = f'./python_code/BMI_{bmi_cat.replace("<", "lt").replace(">", "gt").replace("-", "_")}
            _stacked_distribution.png'
        plt.savefig(filename, dpi=300, bbox_inches="tight")
        plt.close()

        print(f"BMI区间 {bmi_cat} 统计:")
        total_count = len(df_bmi)
        for category in categories:
            count = len(df_bmi[df_bmi["category"] == category])
            percentage = (count / total_count) * 100 if total_count > 0 else 0
            print(f"  {labels[category]}: {count} 人 ({percentage:.1f}%)")
        print(f"  总计: {total_count} 人")
        print()
```

```python


def generate_overall_stacked_bmi_chart():
    df_middle = pd.read_excel("./python_code/bmi_Y_middle_result.xlsx")
    df_cannot_test = pd.read_excel("./python_code/bmi_Y_cannot_test_result.xlsx")
    df_always_can_test = pd.read_excel(
        "./python_code/bmi_Y_always_can_test_result.xlsx"
    )

    df_middle["category"] = "middle"
    df_cannot_test["category"] = "cannot"
    df_always_can_test["category"] = "always_can"

    df_all = pd.concat(
        [df_middle, df_cannot_test, df_always_can_test], ignore_index=True
    )

    def categorize_bmi(bmi):
        if bmi < 30.26:
            return "<30.26"
        elif 30.26 <= bmi < 32.30:
            return "30.26-32.30"
        elif 32.30 <= bmi < 34.92:
            return "32.30-34.92"
        elif 34.92 <= bmi < 39.49:
            return "34.92-39.49"
        else:
            return ">39.49"

    df_all["bmi_category"] = df_all["BMI"].apply(categorize_bmi)

    bmi_categories = ["<30.26", "30.26-32.30", "32.30-34.92", "34.92-39.49", ">39.49"]

    categories = ["cannot", "middle", "always_can"]
    colors = {"cannot": "red", "middle": "yellow", "always_can": "green"}
    labels = {"cannot": "不能达标", "middle": "中间达标", "always_can": "始终达标"}

    fig, ax1 = plt.subplots(figsize=(15, 8))

    all_data_by_category = {cat: [] for cat in categories}

    for bmi_cat in bmi_categories:
        df_bmi = df_all[df_all["bmi_category"] == bmi_cat]

        for category in categories:
            df_category = df_bmi[df_bmi["category"] == category]

            if not df_category.empty:
                if category == "cannot":
                    days = df_category["最晚不达标天数"].tolist()
                elif category == "middle":
                    days = df_category["预测达标天数"].tolist()
                else:
                    days = df_category["最早达标天数"].tolist()

```

```
                labeled_days = [(day, bmi_cat) for day in days]
                all_data_by_category[category].extend(labeled_days)

    x_positions = np.arange(len(bmi_categories))
    bar_width = 0.6

    bottom_values = np.zeros(len(bmi_categories))

    for category in categories:
        heights = []
        for bmi_cat in bmi_categories:
            count = len([x for x in all_data_by_category[category] if x[1] == bmi_cat])
            heights.append(count)

        ax1.bar(
            x_positions,
            heights,
            bar_width,
            bottom=bottom_values,
            label=labels[category],
            color=colors[category],
            alpha=0.7,
            edgecolor="black",
            linewidth=0.5,
        )

        bottom_values = np.add(bottom_values, heights)

    ax1.set_xlabel("BMI区间", fontsize=12)
    ax1.set_ylabel("人数", fontsize=12)
    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)
    ax1.set_xticks(x_positions)
    ax1.set_xticklabels(bmi_categories)
    ax1.legend(fontsize=10)
    ax1.grid(True, alpha=0.3, axis="y")

    plt.tight_layout()

    plt.savefig(
        "./python_code/BMI_all_intervals_stacked_distribution_new.png",
        dpi=300,
        bbox_inches="tight",
    )
    plt.close()

    print("总体堆叠柱状图已生成")


if __name__ == "__main__":
    generate_overall_stacked_bmi_chart()
    generate_stacked_bmi_mixed_charts()
    print("\n图表生成完成！")
```

Listing 7: C(2) 聚类.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    silhouette_score,
    davies_bouldin_score,
    calinski_harabasz_score,
)

plt.rcParams["font.sans-serif"] = ["SimHei"]
plt.rcParams["axes.unicode_minus"] = False


def perform_clustering(df, n_clusters=5):
    X = df[["BMI"]].values
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
    clusters = kmeans.fit_predict(X_scaled)

    df["Cluster"] = clusters

    centers_scaled = kmeans.cluster_centers_
    centers_original = scaler.inverse_transform(centers_scaled)

    return df, kmeans, scaler, centers_original, X_scaled


def evaluate_clustering(X, clusters):
    silhouette_avg = silhouette_score(X, clusters)
    dbi = davies_bouldin_score(X, clusters)
    ch_index = calinski_harabasz_score(X, clusters)

    print("\n聚类效果评估:")
    print(f"轮廓系数 (Silhouette Score): {silhouette_avg:.4f}")
    print(f"DBI指数 (Davies-Bouldin Index): {dbi:.4f}")
    print(f"CH指数 (Calinski-Harabasz Index): {ch_index:.4f}")

    return silhouette_avg, dbi, ch_index


def print_cluster_ranges(df):
    print("\n各聚类的BMI范围:")
    print("-" * 50)
    for i in range(5):
        cluster_data = df[df["Cluster"] == i]["BMI"]
        min_bmi = cluster_data.min()
        max_bmi = cluster_data.max()
        mean_bmi = cluster_data.mean()
        count = len(cluster_data)

        if mean_bmi < 18.5:
```

```
            category = "偏瘦"
        elif mean_bmi < 24:
            category = "正常"
        elif mean_bmi < 28:
            category = "偏胖"
        else:
            category = "肥胖"

        print(
            f"聚类 {i}: {count:3d} 个样本, BMI范围: {min_bmi:.2f}-{max_bmi:.2f}, "
            f"平均值: {mean_bmi:.2f} ({category})"
        )

    print("-" * 50)


def visualize_results(df, centers_original, silhouette_avg, dbi, ch_index):
    fig, axes = plt.subplots(2, 2, figsize=(15, 12))

    for i in range(5):
        cluster_data = df[df["Cluster"] == i]["BMI"]
        axes[0, 0].hist(cluster_data, alpha=0.6, bins=20, label=f"Cluster {i}")

    axes[0, 0].axvline(x=18.5, color="r", linestyle="--", label="偏瘦阈值")
    axes[0, 0].axvline(x=24, color="g", linestyle="--", label="正常阈值")
    axes[0, 0].axvline(x=28, color="orange", linestyle="--", label="偏胖阈值")
    axes[0, 0].set_xlabel("BMI")
    axes[0, 0].set_ylabel("频数")
    axes[0, 0].set_title("各聚类BMI分布")
    axes[0, 0].legend()
    axes[0, 0].grid(True, alpha=0.3)

    cluster_labels = [f"Cluster {i}" for i in range(5)]
    axes[0, 1].bar(
        cluster_labels,
        centers_original.flatten(),
        color=["blue", "green", "red", "purple", "orange"],
    )
    axes[0, 1].set_ylabel("BMI值")
    axes[0, 1].set_title("各聚类中心BMI值")
    axes[0, 1].grid(True, alpha=0.3)

    for i in range(5):
        cluster_data = df[df["Cluster"] == i]
        axes[1, 0].scatter(
            cluster_data["BMI"],
            cluster_data["对应达标天数"],
            alpha=0.6,
            label=f"Cluster {i}",
        )

    axes[1, 0].set_xlabel("BMI")
    axes[1, 0].set_ylabel("达标天数")
    axes[1, 0].set_title("BMI与达标天数的关系")
    axes[1, 0].legend()
```

```
    axes[1, 0].grid(True, alpha=0.3)


def save_results(df, centers_original, silhouette_avg, dbi, ch_index):
    summary_data = {
        "聚类编号": range(5),
        "聚类中心(BMI)": centers_original.flatten(),
        "样本数量": [len(df[df["Cluster"] == i]) for i in range(5)],
        "BMI最小值": [df[df["Cluster"] == i]["BMI"].min() for i in range(5)],
        "BMI最大值": [df[df["Cluster"] == i]["BMI"].max() for i in range(5)],
        "BMI平均值": [df[df["Cluster"] == i]["BMI"].mean() for i in range(5)],
    }
    summary_df = pd.DataFrame(summary_data)

    eval_df = pd.DataFrame(
        {
            "评估指标": ["轮廓系数", "DBI指数", "CH指数"],
            "值": [silhouette_avg, dbi, ch_index],
        }
    )

    with pd.ExcelWriter("聚类分析结果.xlsx") as writer:
        df.to_excel(writer, sheet_name="原始数据与聚类结果", index=False)
        summary_df.to_excel(writer, sheet_name="聚类摘要", index=False)
        eval_df.to_excel(writer, sheet_name="评估指标", index=False)

    print("\n结果已保存到 '聚类分析结果.xlsx'")


def main():
    file_path = "python_code/bmi_Y_result.xlsx"
    df = pd.read_excel(file_path)
    df_result, kmeans, scaler, centers_original, X_scaled = perform_clustering(df)

    silhouette_avg, dbi, ch_index = evaluate_clustering(
        X_scaled, df_result["Cluster"].values
    )

    print_cluster_ranges(df_result)

    visualize_results(df_result, centers_original, silhouette_avg, dbi, ch_index)

    save_results(df_result, centers_original, silhouette_avg, dbi, ch_index)


if __name__ == "__main__":
    main()
```