基于优化算法和相关性分析的乡村农作物种植策略研究

摘　要

随着我国乡村振新计划的推进深入，优化农作物种植策略愈发重要。本文从华北地区某乡村现实情境出发，基于题目所给数据信息，建立数学模型进行分析，从而为该乡村制定未来 7 年的合理种植计划，提高乡村总收入。

针对附件提供数据，依据市场调研报告，进行基本治理，包括数据清洗、填充、转换和补充，在此基础上对数据集进行挖掘为问题求解做准备。

针对问题一，首先在**数据治理**基础上，引入**先验知识**，计算出各种农作物的年预销售量、每亩利润等关键信息。第一问中，将三年内豆类作物轮种作为必要条件，初始化种植方案，建立**动态规划**模型，采用**启发式搜索策略**，优先考虑不重茬情况，将最大化未来 7 年总利润作为目标，求得最终总利润 36953854.63 元；第二问中，将第一问最优种植方案作为初始状态，采用**降价销售**，在第一问基础上，以地块和作物类型为依据，对治理后的数据进行**分块处理**，普通大棚、智慧大棚等分块采用**贪心模式**安排种植，其余分块采用**优化算法**求解，优化过程中，算法遵循种植面积的最小约束、不重茬种植以及豆类作物轮作的全局约束条件，以最大化利润、最小化人力成本和最大化土地利用率作为综合目标，对比**改进遗传算法**和**模拟退火算法**求解出的最优种植方案，最终选择**改进遗传算法**，计算求得总利润 49911805.11 元。

针对问题二，在农作物统计数据的基础上，结合**蒙特卡洛模拟**与 **ARMA 模型**预测部分农作物在预销售量、亩产量、价格等参数上的**增长曲线**。首先，依据农业农村部数据，得到各农作物近两年**关键参数**上的大致趋势，结合随机抽样，得到 2024 和 2025 年的统计数据，以三年农作物统计结果作为**历史数据**，采用 **ARMA 模型**预测 2026~2030 年的**波动数据**。接着，在问题一求解的最优种植方案基础上，引入**智能遗传算法**，以更新后的最大化利润为目标，最终求得总利润 **50701695.09** 元。

针对问题三，引入**相关性分析模型，**挖掘问题二求解出的各农作物发展趋势之间的联系，基于 **Spearman 相关系数**，将结果使用**矩阵热力图**进行可视化展示。为避免**狭窄置信区间**，以 0.97 和-0.97 作为阈值，分别得到最有替代特性的 5 对农作物和最有互补特性的 5 对农作物。最终，模拟数据，与问题二的结果进行比较，证明引入作物之间的相关性，可进一步优化种植方案的结论。

最后本文对所建立的模型进行讨论和分析，综合评价模型的优缺点以及创新点。

关键词： 数据治理 动态规划 启发式搜索 关联关系挖掘

一、问题重述

1.1 问题背景

乡村振兴是我国现代化建设的重要战略之一，而农业产业的可持续发展则是其中的核心内容，对于地处华北山区的许多乡村来说，如何在有限的耕地资源下合理发展有机农业， 提升土地利用效率，已成为亟待解决的现实问题。

该山区地形复杂、气候严苛，耕地被分割成小块，且多为一年一季作物种植。因此，科学选择农作物和优化种植策略至关重要。此外，优化种植策略还需考虑气候变化、市场波动、病虫害等不确定因素，通过合理轮作、避免重茬种植等措施减少风险。同时，精细化管理每种作物的种植面积和分布，降低管理成本。总之，如何优化种植策略就成为了提高农业生产效益的关键。

1.2 问题要求

附件 1 给出了地块名称、类型、面积，以及适宜种植的作物种类和季节。附件 2 提供了 2023 年该乡村农作物种植和相关统计数据，包括种植地块、作物编号、作物名称、作物类型、种植面积、种植季次以及各种作物的亩产量、种植成本和销售单价等详细信息。本文基以上信息建立数学模型解决以下问题：

问题一：

结合相关文献以及附件 1 和 2 求出对应的预期销售量、种植成本、亩产量和销售价格并假定其保持稳定，针对滞销和降价销售两种情况，分别给出 2024-2030 年的最优种植方案。

问题二：

在问题 1 的销售情况下，依据题目中给出的预期销售量、亩产量、种植成本和销售价格存在不确定性的数值，同时以最小化环境影响为目标，综合考虑各种因素，给出 2024-2030 年的最优种植方案。

问题三：

整合两个附件和调研的数据，在问题 2 的基础上，进一步考虑作物之间的可替代性和互补性，以及预期销售量与销售价格、种植成本之间的相关性，给出综合最优种植策略，并进行模拟数据求解与比较分析。

二、问题分析

2.1 问题总体分析

本题从华北山区某乡村现实情境出发，要求设计该乡村未来七年的农作物种植策略，以实现经济和环境的可持续发展。分析过程需考虑乡村现有耕地资源、作物生长规律、市场需求等因素。从数据出发，附件包含 2023 年农作物统计数据、耕地资源统计数据、农作物种植条件统计数据等，需以 2023 年农作物统计数据为基准，对数据进行治理，包含数据清洗、数据处理、数据转换等，提取可用数据；从模型出发，需建立多目标优化模型，优化目标包括最大化销售利润、最大化土地利用率、最小化田间管理成本，具体实施需建立在智能遗传算法、模拟退火算法等优化算法基础上；从目标出发，科学分配农作物种植，可保障农民在面对市场波动、气候突变等因素影响时的基础收入，同时合理的种植顺序可增加土地利用率、提高土壤质量。

2.2 问题一的分析

问题一的目标在于提供 2024-2030 年的最优种植方案。在各种农作物未来的预期销售量、种植成本、亩产量和销售价格相对于 2023 年保持稳定的前提下，本问需对 2023 年农作物统计数据进行充分挖掘，结合前期文献查阅，确定 2024 年的预期销售量、预期销售价格以及利润计算方式。针对总产量超过预期销售量的情况，分为滞销和降价销售两种处理方式。滞销情况下，2024 年各类作物的亩产量应与 2023 年相匹配，同时应结合豆类作物改良土壤、同一地块不能重茬种植等约束，设计合理的种植方案；降价销售情况下，为方便处理，将豆类作物改良土壤、同一地块不能重茬种植设定为必要条件，采用贪心策略，优先选择利润较高的作物，在初始化种植方案后，采用模拟退火和改进的遗传算法进行迭代优化，最终选择未来 7 年总利润较高的种植方案作为最优种植方案。

2.3 问题二的分析

问题二的目标依然在于提供 2024 年-2030 年的最优种植方案。相比问题一，问题二引入更多变量和不确定性因素，包括但不限于销售量的变化、亩产量的变动、部分作物种植成本增长和销售价格波动等。在本问背景下，需引入统计模型，根据历史数据，预估未来 7 年内作物关键参数的变化，应对历史数据较少，通过查阅资料，在蒙特卡洛模型中引入先验知识，求得 2024-2025 年的预测数据，将 2023~2025 年数据作为历史数据，建立自回归移动平均（ARMA）模型，预测 2026 年~2030 年的参数变化，两种统计学模型的相结合，既避免蒙特卡洛随机性过大的局限，又确保自回归移动平均（ARMA）模型具有充足的历史数据。在问题一滞销求得的最优种植方案上，结合预测出的作物每年统计数据，选择遗传算法进行优化，最终求得引入不确定性因素后的最优种植方案。

2.4 问题三的分析

问题三的目标在于挖掘农作物之间的相关性和互补性。相比问题二，问题三引入更多现实因素，在现实生活中，各种农作物之间存在一定的可替代性和互补性。本问需在问题二预测的基础上，充分挖掘各种农作物变化趋势之间的关系。通过查阅资料，本文引入先验知识，农作物品种相近往往存在替代关系，农作物变化趋势相近往往存在互补关系。模型建立方面，使用相关性分析模型，对 2024~2030 年农作物变化趋势进行相关性分析，构建热力图矩阵可视化结果。最终通过模拟数据进行求解，与问题二的结果作比较分析。

整体思路如下所示：

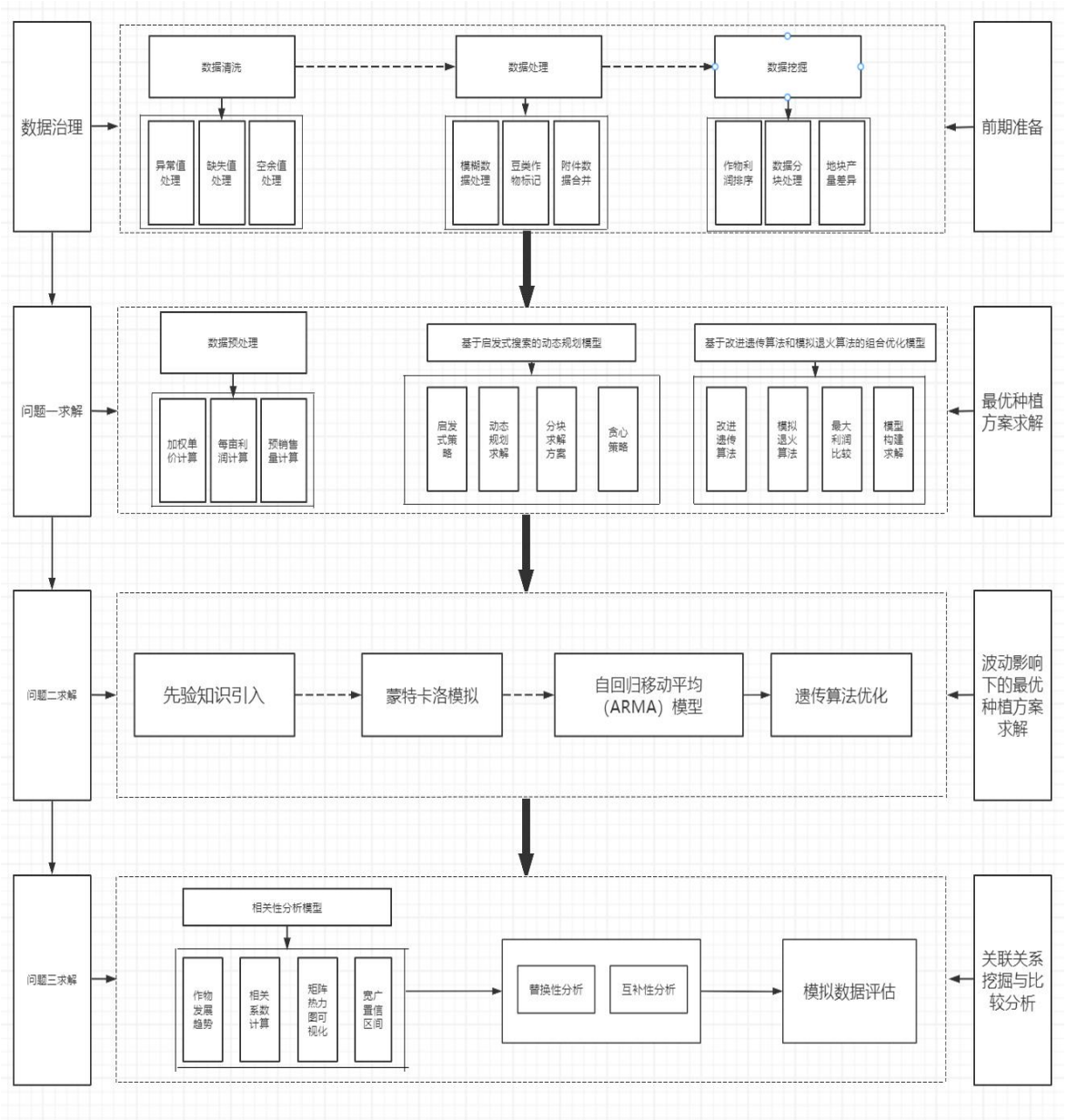


图 2-1 文章思路图

三、数据治理

考虑到数据较为复杂，为更好地分析和理解数据，需在进行问题分析和解答前进行数据整治。根据题目要求，综合附件 1 和附件 2 的数据和信息，并利用 Excel 和 Python 等工具，对数据进行整理和观察以初步了解整体数据的大致情况，便于之后续对问题的分析和解答。考虑附件理论上不存在异常数据，视所有数据均为有效数据。

3.1 模糊数据处理

根据全国农产品批发市场价格行情市场调研报告，农作物销售价格在不同收获季节和需求变化影响的情况下表现出一定程度上的高低波动。查阅资料获得附件中粮食、蔬菜和食用菌三种不同类型作物销售价格的季节性特征，由此分析出权重并计算加权平均售价 Pr，具体公式见 3-1。

$$Pr = \frac{\sum (P_{low} \times W_{low}) + \sum (P_{high} \times W_{high})}{W_{low} + W_{high}} \tag{3-1}$$

其中P_{low}为低价阶段的价格向量，P_{high} 为高价阶段的价格向量，W_{low}为低价阶段的权重向量，W_{high} 为高价阶段的权重向量。

深入调研发现，刘嘉浩在其报告《多期价格权重对农产品市场供给决策的影响和稳定性研究——基于二阶价格加权蛛网模型》的 3.1.1 部分指出，不同种类的农产品在季节性变化和需求波动的影响下，其销售价格的波动性存在显著差异。报告中提供了多种农产品的多期价格权重数据以支持这一观点，并结合附件 2 中提供的农作物销售单价范围，整理出了粮食、蔬菜和菌类这三种不同类型的农产品在价格波动中的高低价段权重计算公式[1]。粮食作物类型的计算公式如 3-2 所示，蔬菜作物类型的计算公式如 3-3 所示，食用菌作物类型的计算公式如 3-3 所示：

$$\mathrm{pr_j} = \frac{7 \times \mathrm{pr_{jmin}} + 5 \times \mathrm{pr_{jmax}}}{12} \tag{3-2}$$

$$\mathrm{pr_j} = \frac{2 \times \mathrm{pr_{jmin}} + 10 \times \mathrm{pr_{jmax}}}{12} \tag{3-3}$$

$$\mathrm{pr_j} = \frac{4 \times \mathrm{pr_{jmin}} + 8 \times \mathrm{pr_{jmax}}}{12} \tag{3-4}$$

3.2 豆类二值化标记

应对土壤改良要求，从 2023 年起要求每个地块（包括大棚）在三年内至少种植一次豆类作物。为了实现这一目标，并便于跟踪和自动化处理，采用二值化方法对豆类作物进行编码标记：即豆类作物被赋予值 1，非豆类作物被赋予值 0。

3.3 附件的合并

附件 1 的“乡村的现有耕地”和“乡村种植的农作物”两张表中有共同的“地块类型”和“种植耕地”字段，匹配得到每块地上种植的作物及其类型。

根据附件 1、2 中作物编号对应，将附件 2 中提供的 2023 年 15 种作物的种植面积、种植季次、亩产量、种植成本、销售单价与附件 1 中数据进行匹配，一方面是作物的基本种植情况和特征，包括种植地块、作物编号、作物名称、作物类型、种植季次、地块类型、是否为豆类、种植面积/亩、亩产量/斤，另一方面是作物的种植经济效益。包括种植成本（元/亩）、加权单价/(元/斤)、每亩利润。实现附件 1 数据信息的补充和完善。

表 3-1 作物的基本种植情况和特征

种植地块	作物编号	作物名称	……	是否为豆类	种植面积/亩	亩产量/斤
A4	1	黄豆	……	1	72	400
B11	1	黄豆	……	1	60	380
C3	1	黄豆	……	1	15	360
B2	2	黑豆	……	1	46	450
B3	3	红豆	……	1	40	380
C6	3	红豆	……	1	20	360
……	……	……	……	……	……	……
E13	41	羊肚菌	……	0	0.6	1000
E14	41	羊肚菌	……	0	0.6	1000
E15	41	羊肚菌	……	0	0.6	1000

表 3-2 作物的种植经济效益

种植地块	作物编号	作物名称	……	种植成本（元/亩）	加权单价(元/斤)	每亩利润/元
A4	1	黄豆	……	400	3.375	950
B11	1	黄豆	……	400	3.375	882.5
C3	1	黄豆	……	400	3.375	815
B2	2	黑豆	……	400	7.666666667	3241.666667
B3	3	红豆	……	350	8.375	2832.5
C6	3	红豆	……	350	8.375	2665
……	……	……	……	……	……	……
E13	41	羊肚菌	……	10000	98.333	88333.333
E14	41	羊肚菌	……	10000	98.333	88333.333
E15	41	羊肚菌	……	10000	98.333	88333.333

通过上述操作，将表 3-1、3-2 导出为一个新的 Excel 文件，即 2023 年农作物统计数据表，详见附录 A。

3.4 数据挖掘与可视化处理

为了探究华北山区乡村不同地块类型的分布情况以及各类农作物的种植效益，基于附件中的数据进行了可视化分析。

1.针对 2023 年农作物统计数据表，挖掘种植平均亩利润较高的作物类型，绘制图 3-1。

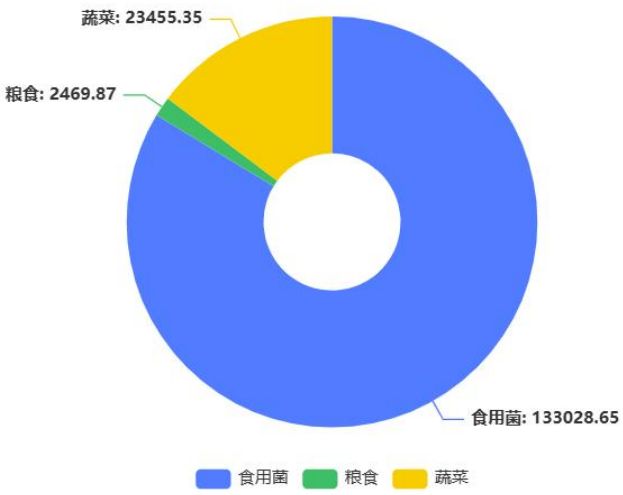


图 3-1 2023 年农作物种植平均亩利润作物类型分布图

根据图 3-1，发现食用菌的平均亩利润最高，种植价值最高。同时食用菌只在普通大棚第二季种植，基于其高经济价值，后续优先考虑每年第二季普通大棚专门用于种植食用菌，再对第一季进行改造。

2.通过柱状图比较平旱地、梯田、山坡地和水浇地的总面积与数量差异展示不同地块类型的分布情况，如图 3-2、3-3 所示。

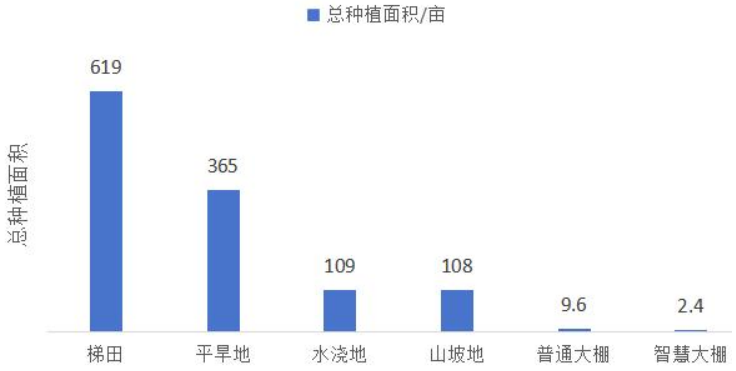


图 3-2 各地块类型总种植面积对比图

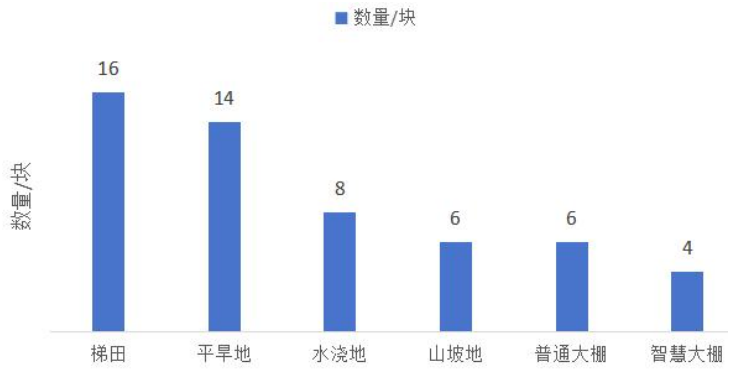


图 3-3 各地块类型总数量对比图

3. 展示不同作物在平旱地、梯田、山坡地、水浇地等类型地块中的亩产量差异，揭示不同地块对作物产量的影响，如图 3-4 所示。

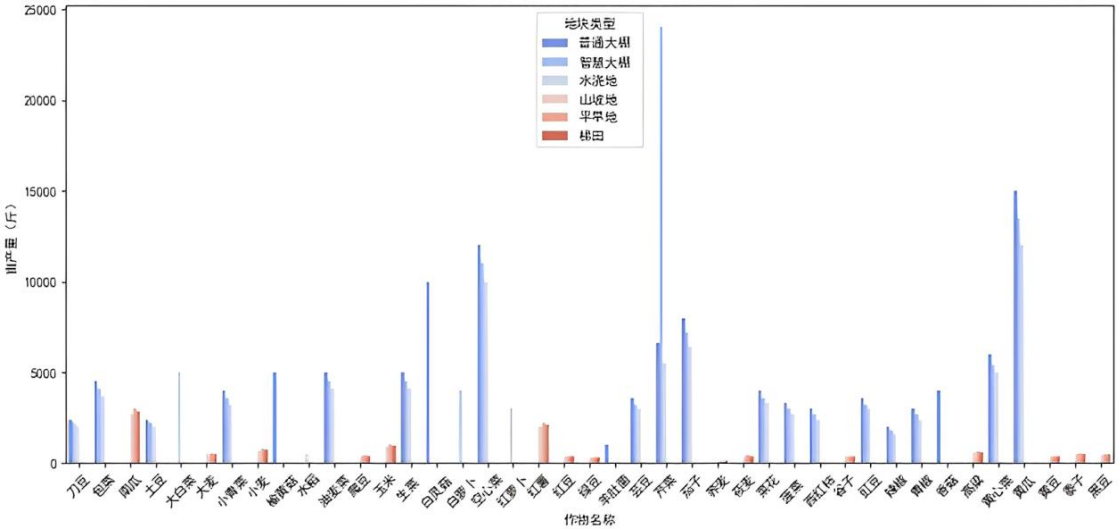


图 3-4 不同作物不同地块类型的亩产量分布图

4. 其他附件数据的可视化结果详见附录 B。

四、模型假设

在构建数学模型以解决华北山区某乡村的农作物种植策略问题时，以下是一些基本假设，这些假设将为模型的建立和求解提供基础：

1. 假设每个季节的种植决策是相对独立的，上一季的种植方案不会直接影响当季的收益（除非涉及轮作约束）。

2. 假设各个地块之间的种植情况互不影响。

3. 乡村的销售空间是固定的，假设其销售空间 2024~2030 年时间内是恒定的。

4. 假设该乡村的耕地总面积和各个地块的面积在研究期间保持不变。

5. 假设所有农作物都能在生长期内完全成熟，且没有极端气候影响。

6. 假设在研究期间，种植技术和田间管理方法保持不变。

7. 假设作物在收获后均能及时销售，不存在滞销问题，除非在模型中特别考虑。

8. 假设作物之间的替代性和互补性可以通过市场调研或历史数据分析得到，并在模型中得到体现。

五、符号说明

符号	说明
T_i	第 i 块地的类型
A_i	第 i 块地的面积
C_j	第 j 种作物类型
S_i	第 j 种作物种植季次
$A_{t,i,j}$	第 $\boldsymbol{t}$ 年第 i 块地第 j 种作物的种植面积
$Y_{i,j}$	第 j 种作物在第 i 块地的亩产量
$Co_{i,j}$	第 j 种作物在第 i 块地的种植成本
Pr_j	第 j 种作物销售价格

六、问题一的模型建立与求解

6.1 数据的预处理

（1）在问题一的假定条件，对数据进行二次处理，获得预期销售量数据；

（2）针对问题一的特性，在第三章数据分块的前提下，对部分数据进行简化，提高模型求解速度，增强代码可读性；

6.1.1 作物预销售量确定

根据问题一获取最优种植方案的目标，需获取作物预销售量，针对该数据缺失，通

过前期调研，张慧在报告《2020—2021 年河南省农作物种子产供需形势分析》的 1.1 章节对供需比进行分析，其中作物总用种量 9.27 万 t，而供需平衡后余种 1.93 万 t，经计算供需比为 79.18%[2][3]。由于河南省与华北地区气候、地形、农业结构、市场经济相似，认为河南省可以代表华北地区的农作物种植供需比。

参考作者提供的供需比，按种植面积的 79.2%计算得到预销售量，具体见表 6-1。

表 6-1 作物预期销售量

种植地块	作物编号	作物名称	……	预销售量（斤）
A4	1	黄豆	……	57.024
B11	1	黄豆	……	47.52
C3	1	黄豆	……	11.88
B2	2	黑豆	……	36.432
B3	3	红豆	……	31.68
C6	3	红豆	……	15.84
……	……	……	……	……
E13	41	羊肚菌	……	0.475
E14	41	羊肚菌	……	0.475
E15	41	羊肚菌	……	0.475

6.1.2 数据分块处理流程

对数据进行分块：Q1 粮食（除水稻外）在平旱地，梯田，山坡地的种植、Q2 蔬菜在大棚（包含智慧大棚），水浇地的种植、Q3 食用菌只在普通大棚第二季的种植。这样分块将问题分解为粮食、蔬菜和菌类三个子问题，可以显著降低解空间的规模，减少计算复杂度，从而提高优化速度。

通过分别处理不同地块类型和作物的种植限制，避免了跨类别的约束冲突，并且简化了约束条件的管理。在这个过程中，每个子问题可以专注于局部目标的优化（三种划分根据背景是互不影响的），例如粮食作物关注产量以及轮番耕种等条件，蔬菜作物注重利润，菌类则无需考虑 3 年必须种植豆类作物的需求。此外，分治策略处理能够灵活利用不同地块的特性，确保种植合理性。同时，它有助于减少田间管理的分散性，提升农业操作效率。

通过这一策略，计算速度得到大幅提升，使得整体种植规划更加高效和可行。

6.1.3 部分种植策略与土壤优化方法

（1）基于 3.4 基础数据挖掘结果，确立 2024~2030 年普通大棚第二季采用同一位置种植同种食用菌的种植策略，既满足方便管理的需求，又满足不重荐种植的要求。

（2）确立统一在 2026、2029 年的第一季进行土壤改造，普通大棚 E1~E16 共 9.6 亩地，使用种植利润最高的刀豆进行改造；智慧大棚 F1~F6 共 3.2 亩地，使用豇豆进行

改造。既能满足销量不造成浪费（刀豆种植在智慧大棚的销量低于预期销售量，即能够全部售出），又能提高利润（豇豆种植在智慧大棚时利润最高）并且方便管理。

6.2 第一问：滞销情况下的最优种植方案求解

6.2.1 基于启发式搜索策略建立动态规划模型

1、采用启发式策略，优先考虑连续 2 年不同种植作物条件，可以快速达到最优状态。

启发式策略是一种基于经验、规则或直觉的方法，旨在通过简化问题、减少搜索空间来快速找到接近最优的解决方案。相比于穷举法等精确算法，启发式策略通常能够在较短时间内获得高质量的可行解，尤其适用于复杂问题。

在题目背景下，启发式策略的搜索方向是连续两年不同种植作物条件。该条件作为优先搜索标准，快速排除不符合条件的作物组合，缩小解的范围，从而更快接近种植效益的最优解。

2、采用动态规划模型，在避免重茬和滞销的前提下，动态规划模型通过选择不同年份、不同作物的种植组合，逐年计算各地块的累计利润，以达到最大化总利润的目标。

动态规划是一种分治法，通过将复杂问题分解为若干个子问题，逐步求解每个子问题，并记录最优解以避免重复计算。

建立状态转移方程步骤如下。

第 *t* 年第 *j* 种作物在第 i 块地上的利润计算公式如 6-1 所示：

$$Pf_{i,j',t} = min(r \times Y_{i,j'} \times A_{t,i,j'}, E_{i,j',t}) \times Pr_{j'} - Co_{i,j'} \times A_{t,i,j'} \quad (6\text{-}1)$$

建立状态转移方程如 6-2 所示：

$$dp_{i,j',t,} = \max_{j \neq j'} (dp_{i,j't,}, dp_{i,j,t-1} + Pf_{i,j',t}) \quad (6\text{-}2)$$

其中，$dp_{i,j,t-1}$表示在第 *t*-1 年第 *j* 种作物在第 *i* 块地的最大累计利润，$dp_{i,j',t}$表示在第 *t* 年第 *j* 种作物在第 *i* 块地的最大累计利润。

6.2.2 目标函数构建

1、全局约束条件：即问题一、二、三均在此条件基础下进行求解

（1）最小种植面积约束

为避免种植过于分散，每种作物在单个大棚上的最小种植面积被设定为 0.3 亩，每种作物在田地的最小种植面积被设定为 1 亩：

$$A_{t,i,j} \geq 0.3,\ i \in [35, 54] \quad (6\text{-}3)$$

$$A_{t,i,j} \geq 1,\ i \in [1, 34] \quad (6\text{-}4)$$

（2）不重茬种植约束

为避免同一种作物连续种植带来的减产情况，每块地在相邻年份不能种植同一种作物：

$$\min(A_{t,i,j}, A_{t-1,i,j}) = 0 \tag{6-5}$$

（3）豆类作物轮作要求

每块地三年内至少要种植一次豆类作物。若某块地在最近三年末种植豆类作物，则该年必须种植豆类:

$$\sum_{t=k-2}^{k} (\sum_{j=1}^{5} A_{t,i,j} + \sum_{j=17}^{19} A_{t,i,j}) > 0 \tag{6-6}$$

2、部分约束条件：即仅问题一情况 1、2 的特殊约束条件

（1）预销售量为 2023 年总产量的 79.2%且保持稳定

$$E_{i,j,t} = E_{i,j,t_0} = r \times Y_{i,j} \times A_{i,j} \tag{6-7}$$

（2）种植成本、亩产量、销售价格保持稳定

3、优化目标：最大化种植总收益，即公式 6-8 所示。

$$\max \sum_{t=1}^{T} \sum_{i=1}^{n} \sum_{j=1}^{m} [min(r \times Y_{i,j} \times A_{t,i,j}, E_{i,j,t}) \times Pr_j - Co_{i,j} \times A_{t,i,j}] \tag{6-8}$$

其中，T 表示总年份，n 表示总地块数，m 表示总作物数。

6.2.3 模型求解结果及分析

基于上述问题建模与求解，将最终求得的最优种植方案填入附件 3 中 Result1_1，并将结果可视化，如图 6-1 所示。

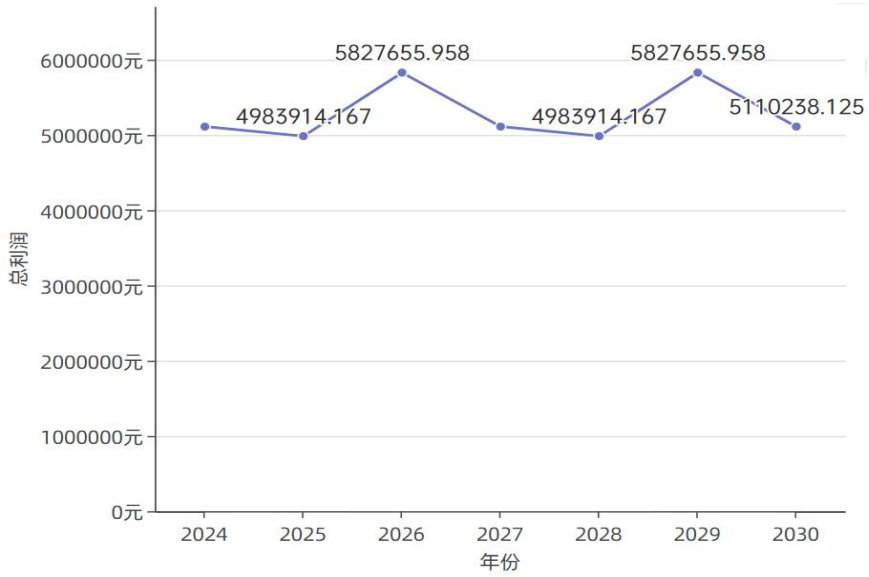


图 6-1 2024-2030 年作物种植总利润

2023 年利润为 5859398.79 元，2024 至 2030 年的总利润为 36953854.63 元，在此期间尽管受到重茬限制与豆类轮作需求的影响，特别是在 2024 年和 2025 年部分高利润作

物的种植面积减少，导致这两年的利润略低于 2023 年，但通过合理的种植结构调整和科学的农作物规划，最终依然成功实现了利润最大化。在整个 7 年规划中，2026 年尤为突出，成为利润相对最高的一年。这不仅是由于 24 年和 25 年轮种限制的解除，使得高利润作物的种植得以恢复，还在于种植结构的灵活调整与优化，确保了利润的回升。尽管 2026 年进行豆类改造，而豆类的相对利润较低，整体的收益依然接近 2023 年的水平，这体现了农作物轮作方案的科学性和可持续性，也展现了在重茬限制下通过精准调整作物种植结构所带来的显著经济效益。最终得出的最高利润，不仅符合农业生产的实际要求，还保证了在限制条件下的合理性与可持续性，使得农业生产收益最大化，充分展现了对利润与可持续发展的双重把控[4][5]。

6.3 第二问：降价销售情况下的最优种植方案求解

6.3.1 基于改进的遗传算法和模拟退火算法的优化组合模型

（1）遗传算法（Genetic Algorithm, GA）是一种基于自然选择和遗传学原理的全局优化搜索算法，通过模拟生物进化过程中的“优胜劣汰”机制，逐步优化问题解。从初始种群开始，遗传算法通过选择、交叉和变异等操作，不断进化，最终得到最优或近似最优解。

- **初始化种群**：随机生成一组初始解，构成初始种群，每个解称为个体，表示为染色体。
- **适应度评估**：计算每个个体的适应度值，衡量解的优劣。
- **选择操作**：根据适应度高低选择下一代的父母，优良基因有更大机会传递给下一代。
- **交叉操作**：通过父母的基因重组生成新的个体（子代）。
- **变异操作**：对子代进行随机基因突变，保持种群多样性，防止陷入局部最优解。
- **生成新种群**：通过选择、交叉和变异产生新种群，替换旧种群。
- **终止条件**：重复以上步骤，直到满足最大迭代次数、找到最优解或种群适应度不再显著提高等条件为止。

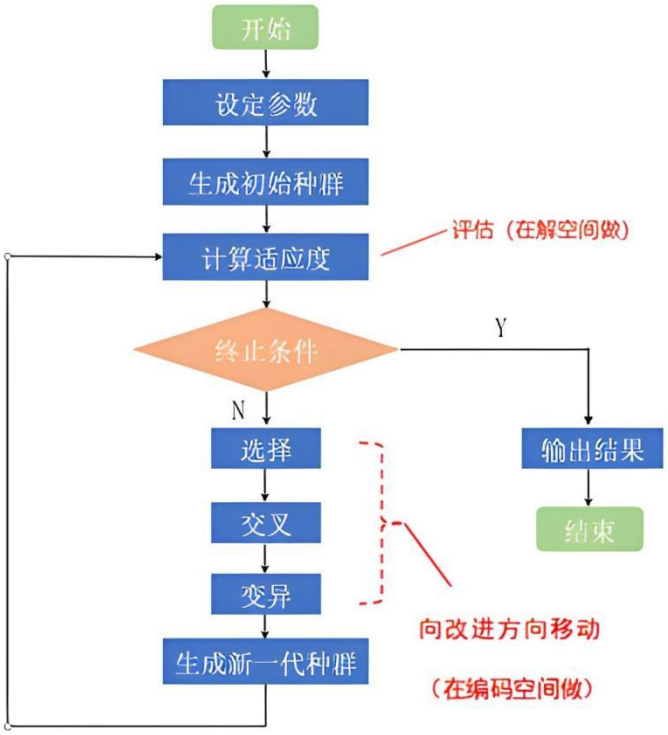


图 6-2 遗传算法流程图

在此基础上对问题进行量化：

- **编码：**将农作物种植方案映射为一组基因，每个基因代表某地块的种植作物。
- **适应度函数：**为目标函数，即总利润，目标是通过选择、交叉、变异操作逐步优化种植方案。
- **初始化种群：**随机生成一批种植方案。
- **适应度评估：**根据适应度函数对每个个体进行评估，计算每个方案的总利润。
- **选择：**采用轮盘选择法，根据适应度高低选择个体。
- **交叉：**交换个体的部分基因片段，即某些地块的种植方案，生成新个体的计算公式如公式 6-9 所示。

$$C = P_x \cdot A + (1 - P_x) \cdot B \tag{6-9}$$

其中P_x为交叉概率，A 与 B 分别为两个不同的种植方案。

- **变异：**随机调整种植面积。
- **终止条件：**达到设定迭代次数或适应度不再提高。

（2）模拟退火算法（Simulated Annealing Algorithm ，SA）核心思想是模仿物理退火过程中的冷却原理，通过接受偶尔的非最优解，避免陷入局部最优，并逐步找到全局最优解。

模拟退火算法应用于作物种植方案优化的应用算法流程，具体见表 6-1。

表 6-1 模拟退化算法流程

模拟退火算法在作物种植方案问题中的应用：

Step1 初始化：随机生成种植方案，计算该解决方案的利润。

Step2 产生新解：在当前解附近的解空间中随机选择一个候选解。

Step3 决策：根据新种植方案的预期利润与当前解决方案比较，若新解的利润高于当前解，则直接接受新解；否则以一定概率接受当前解，概率由 Metropolis 准则当前温度和利润差决定。

$$P = \exp(\frac{\Delta E}{T})$$

其中，ΔE 表示新解与当前解的适应度差（利润差），T 表示当前温度。

Step4 降温：逐渐降低温度，随着温度降低，算法接受差解的概率逐渐减少，使得解逐渐稳定。

Step5 终止：当算法达到设定的迭代次数、温度降至最低利润不再显著提升时，返回最优的种植方案。

在 6.1.3 中使用贪心策略所求的部分种植策略与土壤优化方法，该部分使用贪心已证明是最优解，进而可以降低解空间大小，提高速度，避免搜索路径反复，增大求解效率。从而可以对两种算法进行改进。

6.3.2 模型构建

1、约束条件与 6.2.2 中所述约束条件一致

2、优化目标：最大化种植总收益，最小化环境影响，即公式 6-10 所示。

$$Pf_{i,j',t} = min(r \times Y_{i,j'} \times A_{t,i,j'}, E_{i,j',t}) \times Pr_{j'} - Co_{i,j'} \times A_{t,i,j'}$$

$$\max \sum_{t=1}^{T} \sum_{i=1}^{n} \sum_{j=1}^{m} [Pf_{i,j',t} + \frac{\max(0, r \times Y_{i,j} \times A_{t,i,j} - E_{i,j,t}) \times Pr_j}{2}] \quad (6\text{-}10)$$

6.3.3 模型求解结果

对两种算法的求解结果进行比较，选择求解结果利润最大的算法下的方案作为最优种植方案填入附件 3 中 Result1-2。

遗传算法求解各类作物总利润结果，如下所示：

	粮食	蔬菜	食用菌
利润（元）	16989247.59	25911056.11	7011443.4

退火算法求解各类作物总利润结果，如下所示：

	粮食	蔬菜	食用菌
利润（元）	14156484.06	22679123.14	7011443.4

对比可得遗传算法求得的总利润最大，选择遗传算法求解的最优种植方案。

遗传算法求解下的粮食和蔬菜类作物最大利润七年内变化趋势如图 6-3、6-4 所示，具体优化方案以粮食作物为例，如图 6-5 所示。

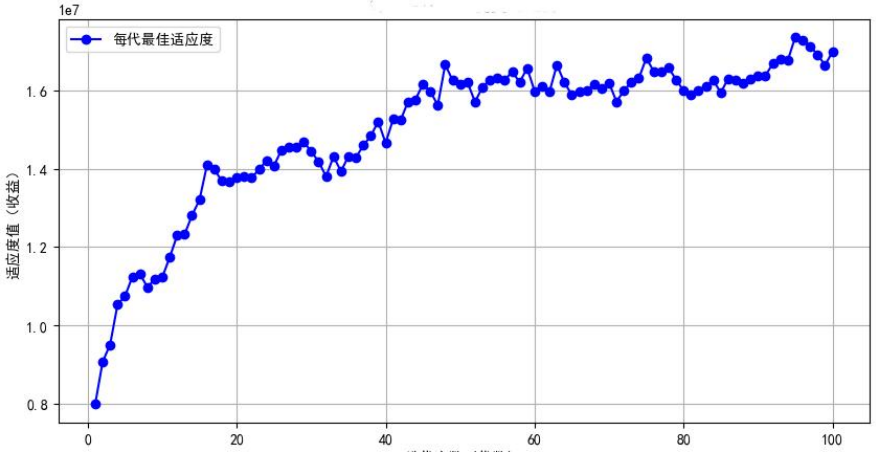


图 6-3 粮食最大利润变化趋势

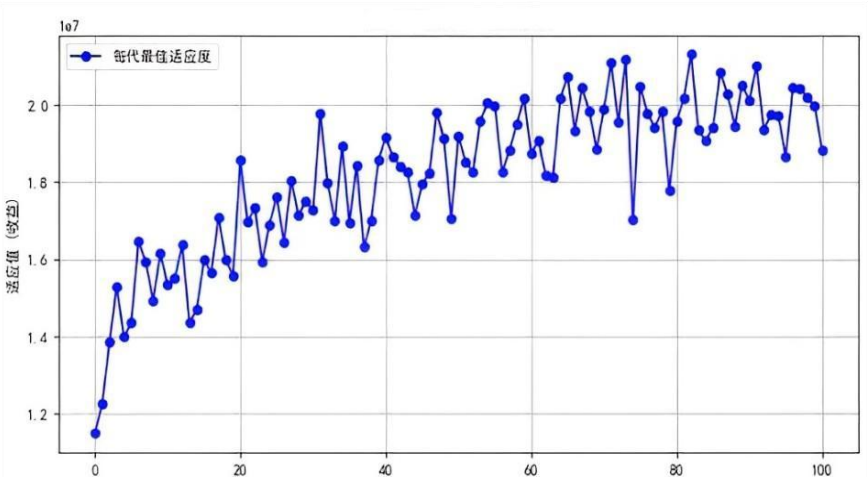


图 6-4 蔬菜最大利润变化趋势

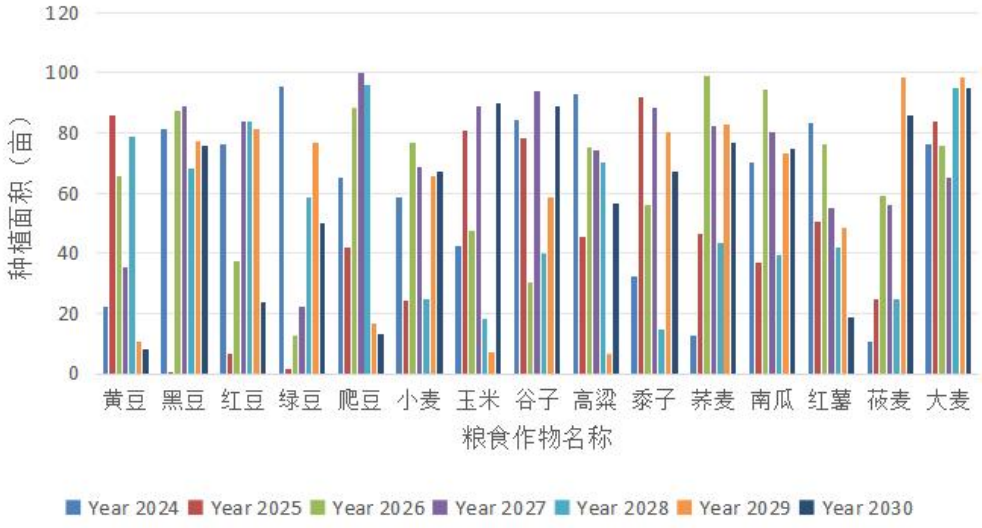


图 6-5 粮食作物具体优化方案

七、 问题二的模型建立与求解

7.1 基于先验知识的蒙特卡洛模型建立

蒙特卡洛模拟（Monte Carlo simulation）是一种基于随机数生成的数学方法，广泛用于处理复杂系统和计算问题，特别是当问题涉及多个变量且充满不确定性时。其核心思想是通过随机抽样，模拟系统中的不同场景，从而近似求解复杂问题。蒙特卡洛模拟的优势在于它不受问题维度或复杂性的限制，适合用于多变量、多条件下的分析与预测。蒙特卡洛模拟依赖于随机抽样，可能导致结果出现较大的波动，因此为了提升模拟的准确性和稳定性，模型引入了基于历史数据的先验知识。通过 2023 年和 2024 年的数据，明确了小麦和玉米的年均销售增长率为 5%至 10%，其他作物的波动在±5%以内，亩产量的年波动为±10%，种植成本平均每年增长 5%。粮食类作物的销售价格较为稳定，蔬菜类作物每年价格上涨约 5%，食用菌价格每年下降 1%至 5%，其中羊肚菌价格下降幅度为 5%。这些先验知识帮助生成更贴近实际的模拟数据，提高了预测的精确度和可信度。

7.2 自回归移动平均（ARMA）模型建立

7.2.1 ARMA 模型简介

自回归滑动平均模型（ARMA 模型）是由自回归模型（AR 模型）和移动平均模型（MA 模型）组合而成的一种时间序列预测模型。ARMA 模型可以表示为 ARMA(p,q)，其中 p 表示自回归项的阶数，q 表示移动平均项的阶数。模型形式可以写成 7-1 所示：

$$x_t = \phi_0 + \phi_1 x_{t-1} + \ldots + \phi_p x_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - \ldots - \theta_q \epsilon_{t-q} \qquad (7-1)$$

其中，x_t是时间序列数据，$\phi_0,\phi_1,\ldots,\phi_p$是自回归系数，$\theta_1, \theta_2, \ldots, \theta_q$是移动平均系数，$\epsilon_t$是误差项。

ARMA 模型的关键在于识别条件、模型阶数的确定以及参数的估计。识别条件通常依赖于时间序列的自相关系数和偏自相关系数的截尾行为。模型阶数可以通过信息准则如 AIC（赤池信息准则）或 BIC（贝叶斯信息准则）来确定，这些准则旨在平衡模型复杂度和拟合优度。参数估计通过最大似然估计来实现。

7.2.2 ARMA 模型的不足与改进

ARMA 模型的精度高度依赖于所使用的时间序列长度。若仅使用 2023 年的数据，时间序列长度过短，可能导致模型对未来预测的置信度较低，容易出现过拟合或偏差。可将 6.1 节处理得到的 2023~2025 年作为历史数据，再使用 ARMA 模型预测得到 2026~2030 年关键参数数据。

7.3 结合蒙特卡洛模型和 ARMA 模型的优势

（1）避免单独采用蒙特卡洛导致的随机性过大

蒙特卡洛模拟依赖随机抽样来生成结果，可能导致结果具有较大波动性，甚至产生不可预期的极端值。通过将蒙特卡洛与 ARMA 模型结合，后者能够通过时间序列数据提供规律性的趋势和波动信息，减少蒙特卡洛模型中因过度随机性导致的结果偏差，从而使整个系统的模拟更加稳定。

（2）引入先验知识保证 2023~2025 年蒙特卡洛模型生成数据的高效性和质量性

蒙特卡洛模型的随机性可以通过引入先验知识来得到优化，确保 2024~2025 的数据更具真实性和代表性，从而在模拟过程中生成更高效且准确的数据，提高整体模拟的可信度。

（3）避免仅使用 2023 年数据导致 ARMA 置信度偏低问题

ARMA 模型的有效性依赖于时间序列的长度，单独使用 2023 年的数据会导致数据过于有限，容易产生过拟合或预测不准确的情况。通过结合蒙特卡洛模型生成的 2023~2025 年多年度数据，ARMA 模型可以利用更长的时间序列数据进行预测，增强其置信度和稳定性，减少由单一数据点引起的偏差。

7.4 目标函数构建

1、在 6.2.2 中所述全局约束条件不变的基础上进行约束条件调整

（1）销售量调整

对于粮食类作物（小麦、玉米）销售量每年增长 5%到 10%：

$$E_{i,j,t} = E_{i,j}(1+\mathrm{u})^{\mathrm{t}}, \mathrm{u} \in [0.05,0.10], \mathrm{j} \in [1,16] \quad (7\text{-}2)$$

对于其他作物波动大约 5%：

$$\mathrm{E_{i,j,t}} = \mathrm{E_{i,j}}(1+\mathrm{u}), \mathrm{u} \in [-0.05,0.05], \mathrm{j} \in [17,41] \quad (7\text{-}3)$$

（2）亩产量调整：所有作物的亩产量每年波动 10%以内

$$\mathrm{Y_{i,j,t}} = \mathrm{Y_{i,j,t_o}}(1+\mathrm{u}), \mathrm{u} \in [-0.05,0.05] \quad (7\text{-}4)$$

（3）种植成本每年增长 5%

$$\mathrm{Co_{i,j,t}} = \mathrm{Co_{i,j,t_0}}(1+0.05)^{\mathrm{t}} \quad (7\text{-}5)$$

（4）蔬菜类作物价格每年上涨 5%，如 7-6 所示；而羊肚菌的价格每年下降 5%，如 7-7 所示；其他食用菌类作物价格每年下降 1%至 5%，如 7-8 所示。

$$\mathrm{Pr_{j,t}} = \mathrm{Pr_{j,t_0}}(1+0.05)^{\mathrm{t}}, \mathrm{j} \in [17,37] \quad (7\text{-}6)$$

$$\mathrm{Pr_{j,t}} = \mathrm{Pr_{j,t_0}}(1-\mathrm{u})^{\mathrm{t}}, \mathrm{u} \in [0.01,0.05], \mathrm{j} \in [38,40] \quad (7\text{-}7)$$

$$\mathrm{Pr_{41,t}} = \mathrm{Pr_{41,t_0}}(1-0.05)^{\mathrm{t}} \quad (7\text{-}8)$$

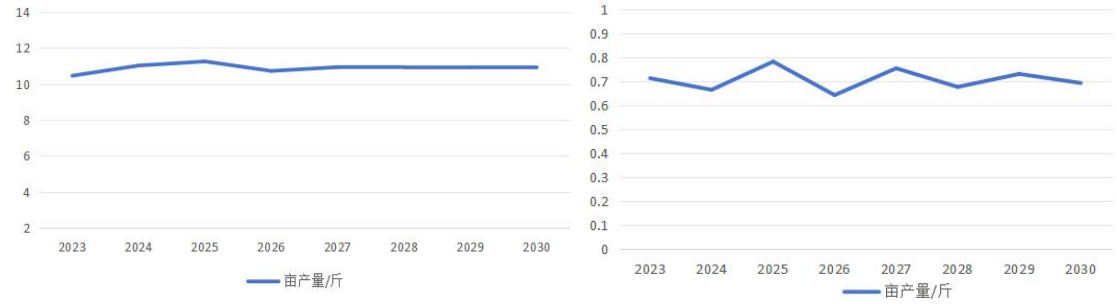
3、优化目标：最大化种植总收益，最小化环境影响，即公式 7-9 所示。

$$Pf_{i,j',t} = min(r \times Y_{i,j'} \times A_{t,i,j'}, E_{i,j',t}) \times Pr_{j'} - Co_{i,j'} \times A_{t,i,j'}$$

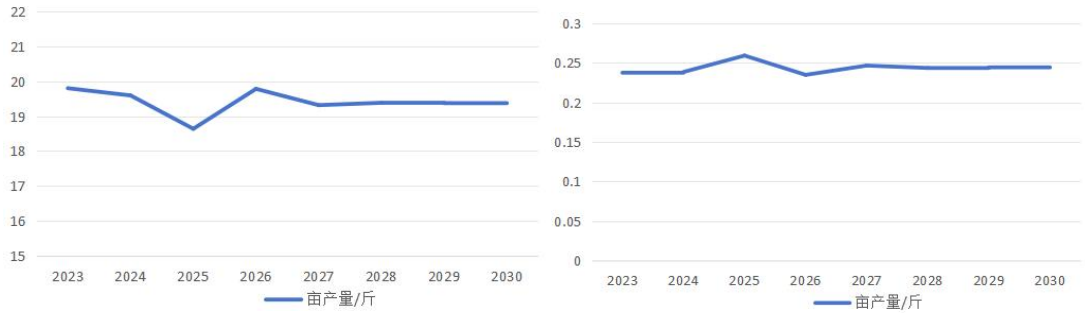
$$\max \sum_{t=1}^{T} \sum_{i=1}^{n} \sum_{j=1}^{m} [Pf_{i,j',t} + \frac{\max(0, r \times Y_{i,j} \times A_{t,i,j} - E_{i,j,t}) \times Pr_j}{2}] \quad (7\text{-}9)$$

7.5 模型求解结果

使用 7.3 节，结合蒙特卡洛和 ARMA 的模型，使用遗传算法计算出 2024~2030 年的农作物统计数据，以刀豆、包菜、白萝卜、黄心菜为例可视化其预测亩产量，如图 7-1 所示，以芸豆、茄子、辣椒、菜花为例可视化其预测销售价格，如图 7-2 所示，其他作物及属性预测见附件 3 中 result2。

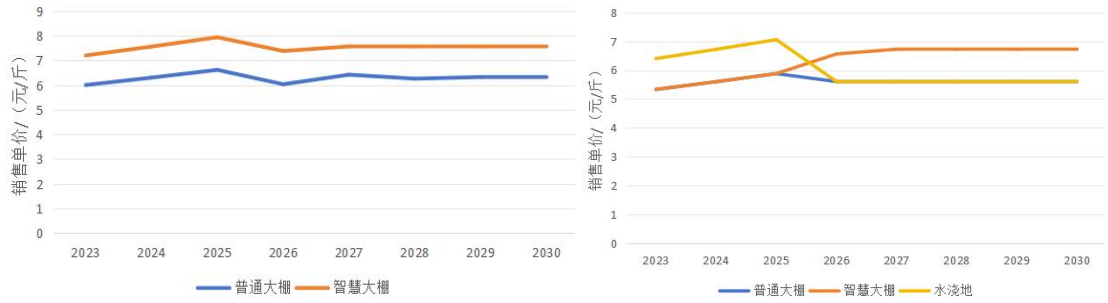


（a）刀豆　　（b）包菜

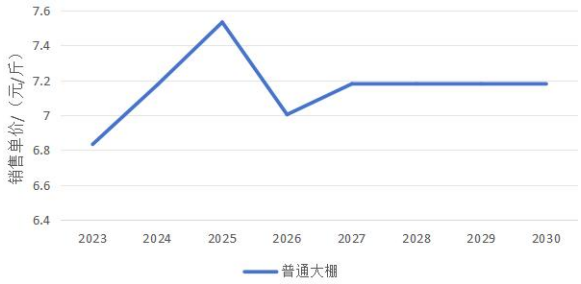


（c）白萝卜　　（d）黄心菜

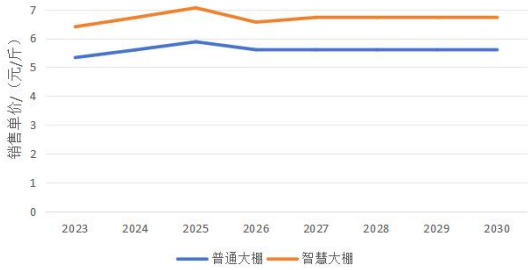
图 7-1 蒙特卡洛和 ARMA 结合下的亩产量预测



（a）芸豆　　（b）茄子



（c）辣椒



（d）菜花

图 7-2 蒙特卡洛和 ARMA 结合下的销售价格预测

为了探究作物产量或销售价格的波动性，并揭示数据的集中趋势与离散程度，利用箱线图来展示数据的分布特征。通过这种方式，我们可以直观地观察到数据的集中区域和分散范围。以刀豆、包菜、白萝卜、黄心菜为例，绘制图 7-3 来展示这些作物亩产量的波动情况；同样地，以芸豆、茄子、辣椒、菜花为例，绘制图 7-4 来展示这些作物销售价格的波动情况。

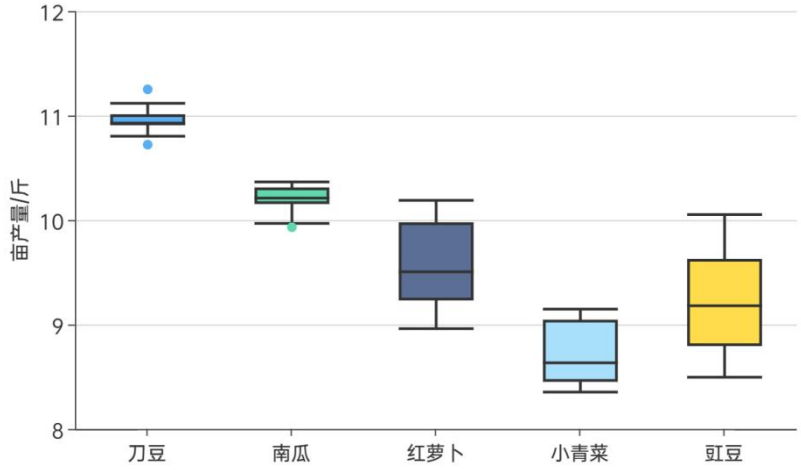


图 7-3 部分作物亩产量预测箱线图

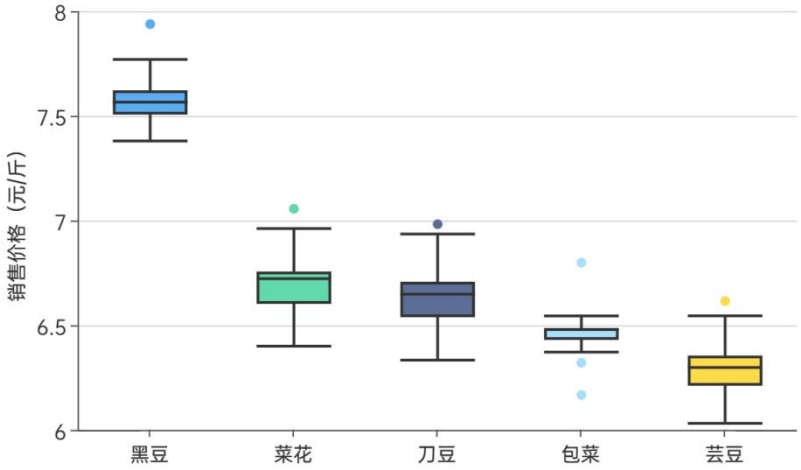


图 7-4 部分作物销售价格预测箱线图

八、问题三模型的建立与求解

8.1 替代性和互补性描述

1、张春琴在其著作《农作物种植结构的模糊随机优化模型与方法》中提出，农作物间的替代性和互补性是由多种因素共同决定的，这些因素共同作用于种植决策，促使农户在特定条件下选择最适宜的作物进行种植，因此，农业种植策略是一个复杂的决策过程，需要综合考虑各种经济、生态和实践因素，以实现农业生产的可持续性[6]。

据附件 A*-*2024~2030 价格预测表，对各农作物变化趋势进行相关性分析，挖掘替代性、互补性，例如，芸豆与豇豆两者均为豆类作物，价格趋势相反，可能存在一定的替代性，青椒与西红柿：在烹饪中，青椒和西红柿常常一同使用因此，它们的价格变化可能表现出一定的互补性。从表格数据也能看出，两者的价格趋势相近。

8.2 相关性模型建立

相关性分析方法有多种，例如 Spearman 相关性分析、Pearson 相关性分析、Kappa 一致性检验等。由于数据是定量的，因此考虑使用 Spearman 和 Pearson 相关性分析。为选择合适方法，首先进行了正态性分布检验。结果显示，预期销售量、销售价格与种植成本不满足正态分布，因此采用 Spearman 相关性分析。

Spearman 相关系数的取值范围为 $[-1,1]$：1 表示完全正单调相关：-1 表示完全负单调相关；0 表示没有显著的单调关系，计算公式如 8-1 所示。

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \tag{8-1}$$

其中，d_i表示两变量的秩次差异，n 表示数据样本的数量。

创建相关矩阵，其中行和列代表不同的变量。对于每一对变量，使用斯皮尔曼公式计算相关系数将相关矩阵作为输入，使用颜色编码来表示相关性的强度，最终输出可视化结果。

8.3 模型求解结果

求解出含有各个农作物的热力图见附录 B，在这里给出 5 种农作物的结果，如图 8-1 所示。

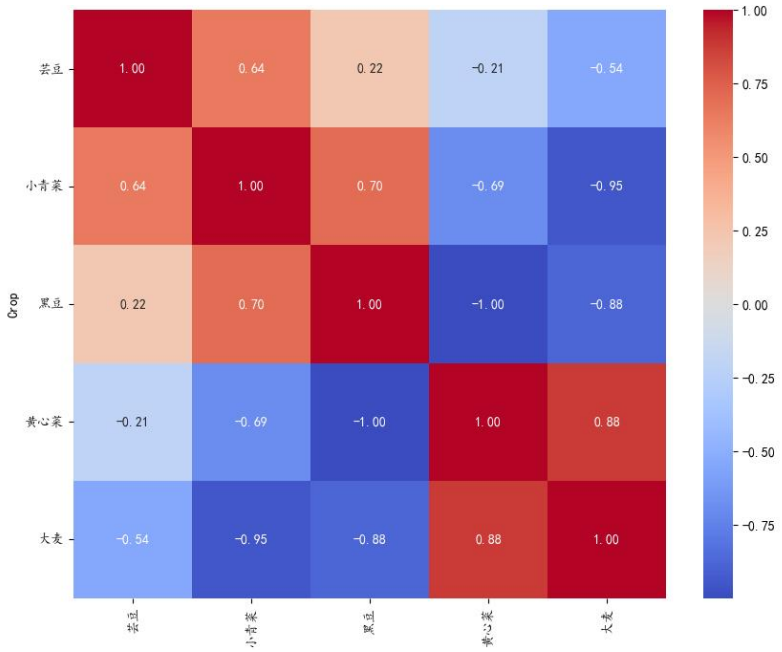


图 8-1 5 种农作物热力图矩阵

然而在本题背景下，相关性系数的绝对值大小并直接反映实际的农业逻辑。例如，在农业种植中，即使两种作物的相关性系数非常高（接近 1 或-1），它们也可能不是完全互补或完全替代的。有时，农民可能会因为管理方便而选择种植某些作物，即使它们在市场上并不是最佳组合。于是以 0.97 作为阈值，寻找小于 0.97 的最大正相关性系数和大于-0.97 的较大负相关性系数，具体结果间表 8-1、8-2。作物之间呈正相关，可互补（$0.97 \leq \rho \leq 1$）的五种农作物热力图，如图 8-2 所示；作物之间呈负相关（$-0.97 \leq \rho \leq 0$）可进行替代的五种农作物热力图，如图 8-3 所示。

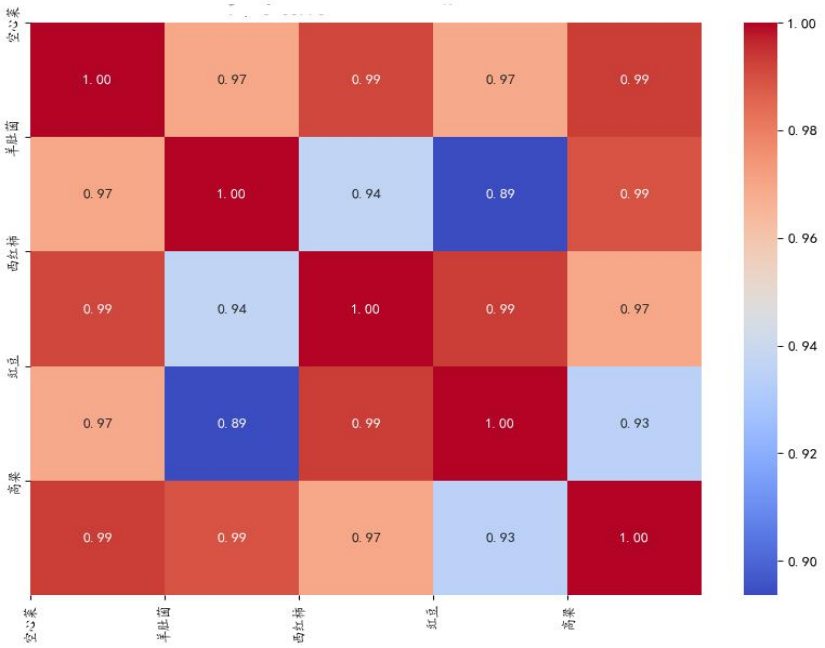


图 8-2 作物之间呈正相关的五种作物（$0.97 \leq \rho \leq 1$）

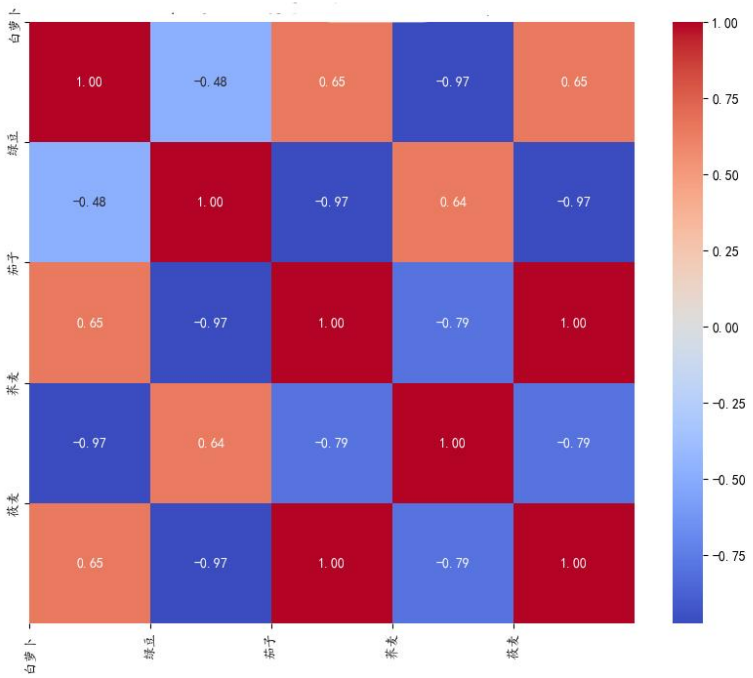


图 8-3 作物之间呈负相关（$-0.97 \leq \rho \leq 0$）的五种农作物热力图

表 8-1 正相关性最高的 5 对农作物

作物 1	作物 2	相关性
空心菜	羊肚菌	0.9697
高粱	西红柿	0.9696
空心菜	豇豆	0.9693

由表 8-1 可知空心菜与羊肚菌、高粱与西红柿、空心菜与豇豆有一定的互补性。空心菜与羊肚菌、豇豆，都能以较高的价格出售，种植者可能会同时种植，以吸引愿意支付高价的消费者；高粱与西红柿的种植成本相似，且可以在同一季节种植，种植者可能会选择同时种植以最大化土地利用。

表 8-2 负相关性最大的 5 对农作物

作物 1	作物 2	相关性
白萝卜	荞麦	-0.9705
茄子	绿豆	-0.9720
绿豆	莜麦	-0.9726

由表 8-2 可知白萝卜与荞麦、绿豆与茄子、绿豆与莜麦有一定的替代性。以白萝卜和荞麦为例，其销售价格相近，消费者可能会在两者之间做出选择，只购买其中的一种，因此在种植时可以进行替代，只种植其中一种。

九、模型的评价与推广

9.1 模型优点

1.数据分块处理将问题分解为粮食、蔬菜和菌类三个子问题，可以显著降低解空间的规模，减少计算复杂度，从而提高优化速度。

2.启发式策略可优先考虑连续 2 年不同种植作物条件，可以快速达到最优状态。

3.动态规划模型能在避免重茬和滞销的前提下动态规划模型通过选择不同年份、不同作物的种植组合，逐年计算各地块的累计利润，以达到最大化总利润的目标。

4.贪心策略在求解部分种植策略与土壤优化方法时，可以降低解空间大小，提高速度，避免搜索路径反复，增大求解效率。从而可以对两种算法进行改进。

5.ARMA 模型与蒙特卡洛结合之后，该模型能够通过时间序列数据提供规律性的趋势和波动信息，减少蒙特卡洛模型中因过度随机性导致的结果偏差，从而使整个系统的模拟更加稳定。

6.蒙特卡洛模型生成的 2023~2025 年多年度数据，使得 ARMA 模型可以利用更长的时间序列数据进行预测，增强其置信度和稳定性，减少由单一数据点引起的偏差。

9.2 模型缺点

1.ARMA 模型的精度高度依赖于所使用的时间序列长度。若仅使用 2023 年的数据，时间序列长度过短，可能导致模型对未来预测的置信度较低，容易出现过拟合或偏差。

2.部分种植策略中，智慧大棚第二季完全采用同一位置种植相同食用菌，简化模型复杂度，忽略了问题二食用菌价格波动变化的影响，可能会对种植策略产生影响。

9.3 模型推广

ARMA 模型能够通过历史数据来识别和模拟时间序列中的模式，从而为未来的预测提供依据，帮助制定决策和策略，可应用于分析电信中的电信号、预测机器人控制系统的行为、恶劣天气事件的预测和疾病的预防管理等多个领域。

十、 参考文献

[1] 刘嘉浩,鞠荣华.多期价格权重对农产品市场供给决策的影响和稳定性研究——基于二阶价格加权蛛网模型[J/OL].价格理论与实践,1-6[2024-09-07].

[2] 张慧,李文丽.2020—2021 年河南省农作物种子产供需形势分析[J].种业导刊,2021,(01):16-18.

[3] 刘景堂.2021 年河南省农作物种子产供需形势分析[J].种业导刊,2022,(01):12-14.

[4] 李六杏,唐立.种植业结构调度的多目标优化模型及 PSO 并行算法[J].重庆科技学院 学报(自然科学版),2020,22(01):60-62.

[5] 鲍树忠.农业种植因素对种植结构的影响及优化策略[J].农业与技术,2020,40(13):90- 91.

[6] 张春琴.农作物种植结构的模糊随机优化模型与方法[J].安徽农业科学,2019,47(18):243-246+250.

附录

附录 A： 2023 年农作物统计数据表

基于问题二 2023~2030 价格预测表

附录 B：不同作物类型与种植季次的种植面积分布图

不同地块类型的种植成本

不同作物的总亩产量分布

所有作物变总热力图矩阵

附录 C：程序代码

附录 A ： 表

2023 年农作物统计数据表

种植地块	作物编号	种植面积（亩）	是否是豆类	加权单价(元/斤)	每亩利润（元）	预售量（亩）	总利润（元）
A1	6	80	0	3.583	2417	63.36	153120
A2	7	55	0	3.083	2583	43.56	112530
A3	7	35	0	3.083	2583	27.72	71610
A4	1	72	1	3.375	950	57.024	54173
A5	4	68	1	7.167	2158	53.856	116239
A6	8	55	0	6.875	2390	43.56	104108
B1	6	60	0	3.583	2273	47.52	108029
B2	2	46	1	7.667	3242	36.432	118100
B3	3	40	1	8.375	2833	31.68	89734
B4	4	28	1	7.167	2015	22.176	44685
B5	5	25	1	6.875	2366	19.8	46839
B6	8	86	0	6.875	2253	68.112	153422
B7	6	55	0	3.583	2273	43.56	99026
B8	8	44	0	6.875	2253	34.848	78495
B9	9	50	0	6.083	3250	39.6	128700
B10	10	25	0	7.667	3473	19.8	68772
B11	1	60	1	3.375	883	47.52	41936
B12	7	45	0	3.083	2429	35.64	86575
B13	14	35	0	5.583	1833	27.72	50820
B14	15	20	0	3.583	1442	15.84	22836

C1	11	15	0	41.667	3817	11.88	45342
C2	12	13	0	1.583	3275	10.296	33719
C3	1	15	1	3.375	815	11.88	9682
C4	13	18	0	3.375	4750	14.256	67716
C5	6	27	0	3.583	2130	21.384	45548
C6	3	20	1	8.375	2665	15.84	42214
D1	20	15	0	3.500	5000	11.88	59400
D1	36	15	0	2.333	8833	11.88	104940
D2	28	10	0	5.500	16000	7.92	126720
D2	35	10	0	2.333	9667	7.92	76560
D3	21	14	0	5.833	12000	11.088	133056
D3	35	14	0	2.333	9667	11.088	107184
D4	22	6	0	5.333	32133	4.752	152698
D4	35	6	0	2.333	9667	4.752	45936
D5	17	10	1	7.667	21000	7.92	166320
D5	36	10	0	2.333	8833	7.92	69960
D6	18	12	1	6.333	11667	9.504	110880
D6	37	12	0	3.000	8500	9.504	80784
D7	16	22	0	7.167	2903	17.424	50588
D8	16	20	0	7.167	2903	15.84	45989
E1	18	0.6	1	6.333	14000	0.4752	6653
E1	38	0.6	0	56.875	281375	0.4752	133709
E2	24	0.6	0	4.833	12500	0.4752	5940
E2	38	0.6	0	56.875	281375	0.4752	133709
E3	25	0.6	0	5.333	18333	0.4752	8712
E3	38	0.6	0	56.875	281375	0.4752	133709
E4	26	0.6	0	6.167	24250	0.4752	11524
E4	39	0.6	0	18.917	73667	0.4752	35006
E5	28	0.6	0	5.500	20000	0.4752	9504
E5	39	0.6	0	18.917	73667	0.4752	35006
E6	27	0.6	0	4.667	21333	0.4752	10138
E6	39	0.6	0	18.917	73667	0.4752	35006
E7	19	0.6	1	6.000	19200	0.4752	9124
E7	40	0.6	0	15.833	148333	0.4752	70488

E8	19	0.6	1	6.000	19200	0.4752	9124
E8	40	0.6	0	15.833	148333	0.4752	70488
E9	18	0.6	1	6.333	14000	0.4752	6653
E9	40	0.6	0	15.833	148333	0.4752	70488
E10	17	0.6	1	7.667	25200	0.4752	11975
E10	41	0.6	0	98.333	88333	0.4752	41976
E11	17	0.6	1	7.667	25200	0.4752	11975
E11	41	0.6	0	98.333	88333	0.4752	41976
E12	22	0.6	0	5.333	40267	0.4752	19135
E12	41	0.6	0	98.333	88333	0.4752	41976
E13	21	0.6	0	5.833	15100	0.4752	7176
E13	41	0.6	0	98.333	88333	0.4752	41976
E14	29	0.6	0	6.667	96500	0.4752	45857
E14	41	0.6	0	98.333	88333	0.4752	41976
E15	30	0.3	0	5.000	23000	0.2376	5465
E15	27	0.3	0	4.667	21333	0.2376	5069
E15	41	0.6	0	98.333	88333	0.4752	41976
E16	31	0.6	0	6.833	12467	0.4752	5924
E16	41	0.6	0	98.333	88333	0.4752	41976
F1	32	0.3	0	4.800	47300	0.2376	11238
F1	33	0.3	0	5.200	25330	0.2376	6018
F1	24	0.3	0	6.467	15260	0.2376	3626
F1	21	0.3	0	7.000	16260	0.2376	3863
F2	25	0.3	0	6.400	19740	0.2376	4690
F2	26	0.3	0	7.400	26490	0.2376	6294
F2	22	0.3	0	6.400	43440	0.2376	10321
F2	29	0.3	0	8.000	104150	0.2376	24746
F3	17	0.6	1	9.200	26800	0.4752	12735
F3	28	0.3	0	6.600	21560	0.2376	5123
F3	30	0.3	0	6.000	24800	0.2376	5892
F4	19	0.6	1	7.200	20400	0.4752	9694
F4	34	0.3	0	4.467	25600	0.2376	6083
F4	23	0.3	0	6.533	16600	0.2376	3944

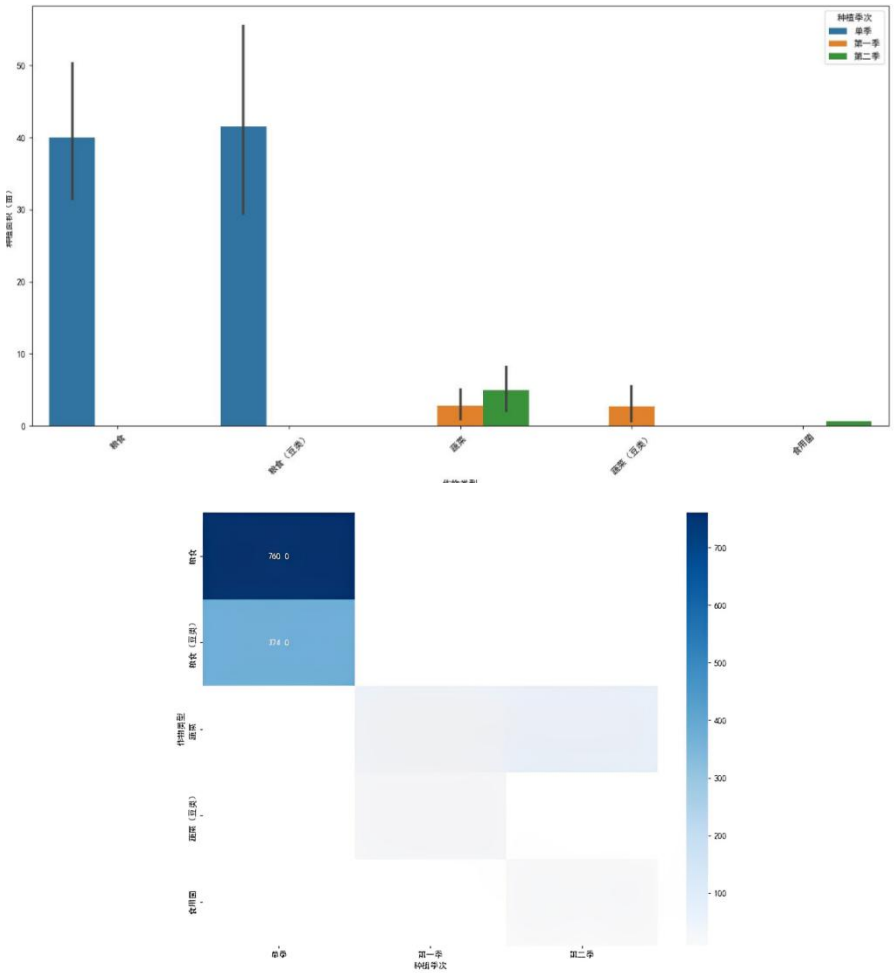
基于问题二 2023~2030 价格预测表：

作物名称	地块类型	2023 价格	2024 价格	2025 价格	2026 价格	2027 价格	2028 价格	2029 价格	2030 价格
芸豆	普通大棚	6.00	6.30	6.62	6.03	6.42	6.26	6.32	6.30
芸豆	智慧大棚	7.20	7.56	7.94	7.38	7.57	7.57	7.57	7.57
玉米	平旱地	3.08	3.08	3.08	3.08	3.08	3.08	3.08	3.08
玉米	梯田	3.08	3.08	3.08	3.08	3.08	3.08	3.08	3.08
榆黄菇	普通大棚	56.88	54.67	52.54	56.72	53.75	55.14	54.49	54.79
莜麦	梯田	5.58	5.58	5.58	5.58	5.58	5.58	5.58	5.58
油麦菜	普通大棚	4.67	4.90	5.15	4.69	4.99	4.87	4.92	4.90
羊肚菌	普通大棚	98.33	93.42	88.75	98.19	91.18	94.64	92.93	93.78
小青菜	普通大棚	5.50	5.78	6.06	5.78	5.78	5.78	5.78	5.78
小青菜	智慧大棚	5.50	5.78	6.06	6.76	6.93	6.93	6.93	6.93
小青菜	水浇地	6.60	6.93	7.28	5.78	5.78	5.78	5.78	5.78
小麦	山坡地	3.58	3.58	3.58	3.58	3.58	3.58	3.58	3.58
小麦	平旱地	3.58	3.58	3.58	3.58	3.58	3.58	3.58	3.58
小麦	梯田	3.58	3.58	3.58	3.58	3.58	3.58	3.58	3.58
香菇	普通大棚	18.92	18.67	18.43	18.90	18.56	18.72	18.65	18.68
西红柿	普通大棚	5.83	6.13	6.43	6.13	6.13	6.13	6.13	6.13
西红柿	智慧大棚	5.83	6.13	6.43	7.17	7.36	7.36	7.36	7.36
西红柿	水浇地	7.00	7.35	7.72	6.13	6.13	6.13	6.13	6.13
土豆	水浇地	3.50	3.68	3.86	3.68	3.68	3.68	3.68	3.68
水稻	水浇地	7.17	7.17	7.17	7.17	7.17	7.17	7.17	7.17
黍子	梯田	7.67	7.67	7.67	7.67	7.67	7.67	7.67	7.67
生菜	普通大棚	5.00	10.25	21.55	7.25	12.30	12.04	12.05	12.05
生菜	智慧大棚	6.00	12.30	25.86	8.70	14.75	14.44	14.46	14.46
青椒	普通大棚	4.83	5.08	5.33	5.08	5.08	5.08	5.08	5.08
青椒	智慧大棚	6.47	6.79	7.13	6.63	6.79	6.79	6.79	6.79
芹菜	智慧大棚	4.47	4.69	4.92	4.69	4.69	4.69	4.69	4.69
茄子	普通大棚	5.33	5.60	5.88	5.60	5.60	5.60	5.60	5.60
茄子	智慧大棚	5.33	5.60	5.88	6.56	6.72	6.72	6.72	6.72
茄子	水浇地	6.40	6.72	7.06	5.60	5.60	5.60	5.60	5.60
荞麦	山坡地	41.67	41.67	41.67	41.67	41.67	41.67	41.67	41.67
爬豆	梯田	6.88	7.22	7.96	7.02	7.35	7.34	7.34	7.34

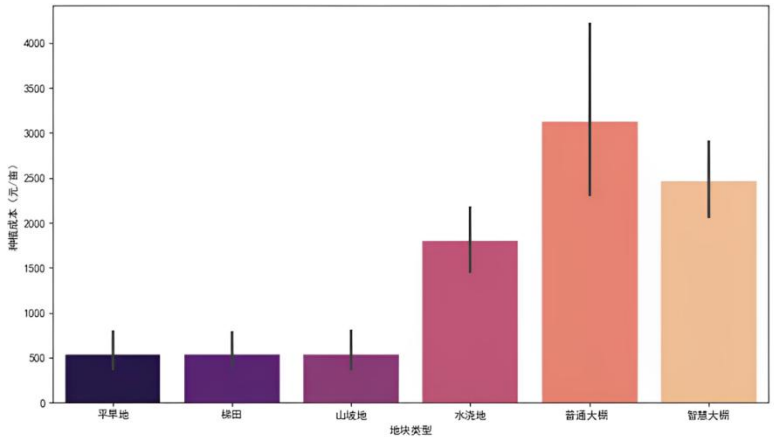
南瓜	山坡地	1.58	1.66	1.75	1.66	1.66	1.66	1.66	1.66
绿豆	平旱地	7.17	7.53	8.30	7.32	7.66	7.65	7.65	7.65
绿豆	梯田	7.17	7.53	8.30	7.32	7.66	7.65	7.65	7.65
辣椒	普通大棚	6.83	7.18	7.53	7.00	7.18	7.18	7.18	7.18
空心菜	智慧大棚	4.80	5.04	5.29	5.04	5.04	5.04	5.04	5.04
豇豆	普通大棚	7.67	8.05	8.45	7.71	8.20	7.99	8.08	8.05
豇豆	智慧大棚	7.67	8.05	8.45	9.43	9.67	9.67	9.67	9.67
豇豆	水浇地	9.20	9.66	10.14	7.86	8.06	8.06	8.06	8.06
黄心菜	智慧大棚	5.20	5.46	5.73	5.46	5.46	5.46	5.46	5.46
黄瓜	普通大棚	6.67	7.00	7.35	6.83	7.01	7.00	7.00	7.00
黄瓜	智慧大棚	8.00	8.40	8.82	8.20	8.41	8.41	8.41	8.41
黄豆	山坡地	3.38	3.54	3.91	3.45	3.61	3.60	3.60	3.60
黄豆	平旱地	3.38	3.54	3.91	3.45	3.61	3.60	3.60	3.60
黄豆	梯田	3.38	3.54	3.91	3.45	3.61	3.60	3.60	3.60
红薯	山坡地	3.38	3.54	3.72	3.55	3.55	3.55	3.55	3.55
红萝卜	水浇地	3.00	3.15	3.31	3.15	3.15	3.15	3.15	3.15
红豆	山坡地	8.38	8.79	9.70	8.55	8.96	8.94	8.94	8.94
红豆	梯田	8.38	8.79	9.70	8.55	8.96	8.94	8.94	8.94
黑豆	梯田	7.67	8.05	8.88	7.83	8.20	8.18	8.18	8.18
谷子	平旱地	6.88	6.88	6.88	6.87	6.87	6.87	6.87	6.87
谷子	梯田	6.88	6.88	6.88	6.87	6.87	6.87	6.87	6.87
高粱	梯田	6.08	6.08	6.08	6.08	6.08	6.08	6.08	6.08
刀豆	普通大棚	6.33	6.65	6.98	6.37	6.78	6.60	6.68	6.65
刀豆	水浇地	6.33	6.65	6.98	6.49	6.65	6.65	6.65	6.65
大麦	梯田	3.58	3.58	3.58	3.58	3.58	3.58	3.58	3.58
大白菜	水浇地	2.33	2.45	2.57	2.34	2.50	2.43	2.46	2.45
菜花	普通大棚	5.33	5.60	5.88	5.60	5.60	5.60	5.60	5.60
菜花	智慧大棚	6.40	6.72	7.06	6.56	6.72	6.72	6.72	6.72
菠菜	智慧大棚	6.53	6.86	7.56	6.67	6.99	6.97	6.97	6.97
包菜	普通大棚	6.17	6.48	6.80	6.32	6.48	6.48	6.48	6.48
包菜	智慧大棚	7.40	7.77	8.16	7.58	7.78	7.78	7.78	7.78
白萝卜	水浇地	2.33	2.45	2.57	2.35	2.50	2.43	2.46	2.45
白灵菇	普通大棚	15.83	15.38	14.94	15.80	15.19	15.48	15.34	15.41

附录 B ： 图

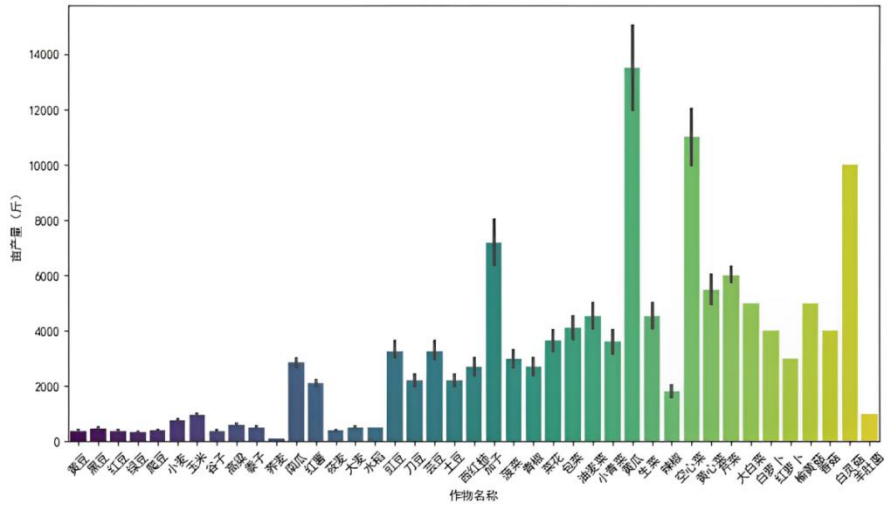
不同作物类型与种植季次的种植面积分布：展示不同作物在各个季次中的种植面积，结合热力图进一步突出高频种植作物及其在不同时期的面积分布。



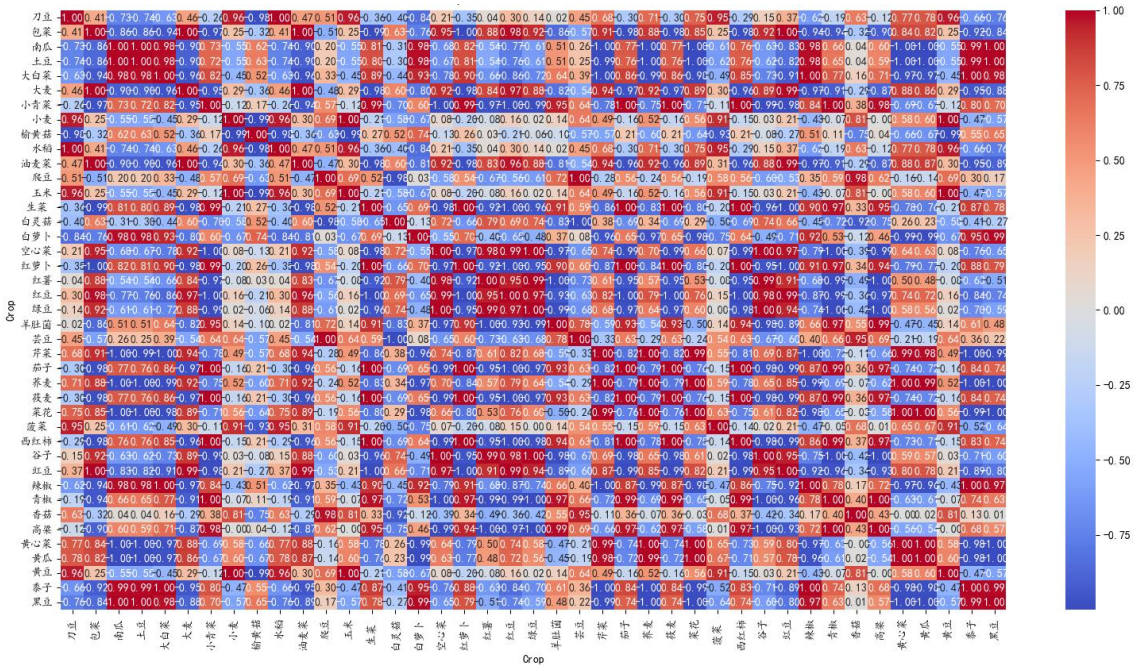
不同地块类型的种植成本：展示平旱地、梯田、山坡地和水浇地等不同类型地块的种植成本。



不同作物的总亩产量分布：



所有作物变总热力图矩阵：



附录 C ： 相关代码

相关性分析

```
import pandas as pd
import seaborn as sns
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['KaiTi']
plt.rcParams['axes.unicode_minus']=False
import numpy as np

# 读取 Excel 文件
```

```
file_path = '亩产量预测.xlsx'  # 请根据实际文件路径修改
df = pd.read_excel(file_path, index_col='Crop')

# 转置 DataFrame，使作物成为行，年份成为列
df = df.T

# 计算相关系数矩阵
correlation_matrix = df.corr()

# 找出负相关性最大的农作物对，且相关性小于-0.97
# 先将矩阵 flatten 为一个序列，然后找出满足条件的值
flat_corr = correlation_matrix.unstack()
negative_corr_pairs = flat_corr[(flat_corr < -0.97)].dropna()

# 按相关性强度（绝对值）排序
negative_corr_pairs = negative_corr_pairs.sort_values(ascending=True, key=abs)

# 选择负相关性最大的 5 对农作物
top_5_negative_pairs = negative_corr_pairs.head(5)

# 打印负相关性最大的 5 对农作物
print("负相关性最大的 5 对农作物:")
for index, value in top_5_negative_pairs.items():
print(f"{index[0]} 和 {index[1]} 的相关性为 {value:.4f}")

# 绘制热力图
selected_crops = np.unique(top_5_negative_pairs.index.get_level_values(0))
selected_crops = np.append(selected_crops, top_5_negative_pairs.index.get_leve
l_values(1))
selected_crops = np.unique(selected_crops)
selected_crops = list(selected_crops)  # 转换为列表

plt.figure(figsize=(10, 8))
ax = sns.heatmap(correlation_matrix.loc[selected_crops, selected_crops], annot
=True, cmap='coolwarm', fmt=".2f")
plt.title('Top 5 Negative Correlation Crops Heatmap')
# 设置 x 轴和 y 轴的标签
plt.xticks(range(len(selected_crops)), selected_crops, rotation=90)
plt.yticks(range(len(selected_crops)), selected_crops)
```

每亩利润求解

```
import pandas as pd

# 读取 Excel 文件
file_path = '加权单价_2023 年统计的相关数据.xlsx'
df = pd.read_excel(file_path)

# 计算每亩利润
df['每亩利润'] = (df['加权单价/(元/斤)'] * df['亩产量/斤']) - df['种植成本/(元/亩)']

# 将结果保存到新的 Excel 文件
output_path = '每亩利润_2023 年统计的相关数据.xlsx'
df.to_excel(output_path, index=False)
```

获取最低最高价格

```
import pandas as pd

# 读取 Excel 文件
df = pd.read_excel('作物类型_2023 年统计的相关数据.xlsx')  # 替换 'your_file.xlsx' 为你的文件名

# 定义一个函数来处理价格区间
def split_price(price_range):
    # 分割价格区间并返回最低和最高价格
    prices = price_range.split('-')
    return float(prices[0]), float(prices[1])

# 应用函数并创建新的列
df[['最低价格', '最高价格']] = df.apply(lambda row: split_price(row['销售单价/(元/斤)']), axis=1, result_type='expand')

# 保存更新后的表格到新的 Excel 文件
df.to_excel('最低最高单价_2023 年统计的相关数据.xlsx', index=False)
```

总预销售量求解

```
import pandas as pd

# 读取 Excel 文件
df = pd.read_excel('2023 年农作物各类数据最终版 1.1.xlsx')  # 替换 'your_file.xlsx' 为你的文件名
```

```
# 打印原始数据（可选）
print("原始数据:")
print(df)

# 统计每种作物名称的总预售量（亩）
total 预售量 = df.groupby('作物名称')['预售量（亩）'].sum()

# 打印统计结果
print("每种作物名称的总预售量（亩）:")
print(total 预售量)

# 将结果保存到新的 Excel 文件
total 预售量.to_excel('总预售量_2023 年农作物各类数据最终版 1.1.xlsx')
```

平均价格求解

```
import pandas as pd

# 读取 Excel 文件
file_path = '加权单价_2023 年统计的相关数据.xlsx'
df = pd.read_excel(file_path)

# 计算销售单价的平均值
def calculate_average_price(price_range):
    low, high = map(float, price_range.split('-'))
    return (low + high) / 2

# 应用函数计算平均值
df['销售单价/(元/斤)'] = df['销售单价/(元/斤)'].apply(calculate_average_price)

# 将修改后的数据保存到新的 Excel 文件
output_path = '平均价格_2023 年统计的相关数据.xlsx'  # 您可以自定义输出文件的名称
df.to_excel(output_path, index=False)  # index=False 表示不保存行索引到文件
```

加权单价求解

```
import pandas as pd

# 读取 Excel 文件
df = pd.read_excel('wgx_23 年农作物价格表.xlsx')  # 替换 'your_file.xlsx' 为你的文件名
```

```python
# 定义一个函数来计算加权单价
def calculate_weighted_price(row):
    if row['作物类型'] == '粮食' or row['作物类型'] == '粮食（豆类）':
        return 4 * (row['最低单价'] + row['平均单价']) + 8 * (row['平均单价
'] + row['最高单价'])
    elif row['作物类型'] == '蔬菜' or row['作物类型'] == '蔬菜（豆类）':
        return 10 * (row['最低单价'] + row['平均单价']) + 2 * (row['平均单价
'] + row['最高单价'])
    elif row['作物类型'] == '食用菌':
        return 7 * (row['最低单价'] + row['平均单价']) + 5 * (row['平均单价
'] + row['最高单价'])
    else:
        return None  # 如果作物类型不匹配，返回 None

# 应用函数计算加权单价
df['加权单价/(元/斤)'] = df.apply(calculate_weighted_price, axis=1)

df['加权单价/(元/斤)'] = df['加权单价/(元/斤)']*1.0/24
# 保存更新后的表格到新的 Excel 文件
df.to_excel('wgx_加权单价_2023 年统计的相关数据.xlsx', index=False)
```

优化问题一情况一的种植方案

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


# 读取 Excel 文件
def read_excel(file_path):
    return pd.read_excel(file_path, index_col=0)


# 写入 Excel 文件
def write_excel(dataframe, file_path):
    dataframe.to_excel(file_path, index=True)


# 生成 2024 年的数据
def generate_2024_data(historic_data):
    n_rows, n_cols = historic_data.shape
    new_year_data = historic_data.copy()
```

```
    # 遍历每个地块
    for row_index in range(1, n_rows + 1):
        current_crops = historic_data.iloc[row_index - 1, 1:].values
        new_crops = [0] * (n_cols - 1)  # 初始化新作物数组

        # 尝试为当前地块找到一个新的作物组合
        for col_index in range(n_cols - 1):
            if current_crops[col_index] > 0:
                # 避免在同一地块上种植相同的作物
                possible_crops = [crop for crop in range(1, n_cols + 1) if crop
 != current_crops[col_index]]
                new_crops[col_index] = possible_crops[np.random.choice(len(pos
sible_crops))]  # 随机选择一个不同的作物

        # 更新数据
        new_year_data.iloc[row_index - 1, 1:] = new_crops

    return new_year_data

# 主函数
def main():
    file_path = '2023 年 ABC 地块数据.xlsx'
    data_2023 = read_excel(file_path)
    data_2024 = generate_2024_data(data_2023)
    output_file_path = '2024 年 ABC 地块数据.xlsx'
    write_excel(data_2024, output_file_path)


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

亩产量预测

```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import numpy as np

# 读取数据
df = pd.read_csv('最终价格_2024_2025.csv')

# 初始化一个空的 DataFrame 来存储预测结果，确保列名与原始数据一致（这里假设有作物名称、地块类型和 2023-2030 年的价格列）
predictions = pd.DataFrame(columns=['作物名称', '地块类型'] + [f'{year}价格
```

```
' for year in range(2023, 2031)])

# 对每种作物和地块类型进行分组，并预测价格
grouped = df.groupby(['作物名称', '地块类型'])
for (crop, land_type), group in grouped:
    historical_prices = group[['2023 价格', '2024 价格', '2025 价格
']].values.flatten()  # 将价格数据展平为一维数组

    # 检查数据是否包含 NaN 值
    if np.isnan(historical_prices).any():
        continue  # 跳过包含 NaN 的数据组

    # 尝试使用 ARIMA 模型进行预测（这里使用简单的 ARIMA(1,0,1)作为示例）
    try:
        model = ARIMA(historical_prices, order=(1, 0, 1))
        model_fit = model.fit()
        forecast = model_fit.forecast(steps=5)  # 预测未来的 5 个价格点

        # 创建新的行来存储预测结果
        new_row = [crop, land_type] + [np.nan] * 3  # 2023-2025 年的价格已经是
历史数据，所以这里填 NaN（或者可以是实际值）
        new_row += list(forecast)  # 添加预测的价格
        new_row += [np.nan] * (2030 - 2025 - 1 - len(forecast))  # 填充剩余年
份的 NaN（如果有的话）

        # 使用 loc 来添加新行到 predictions DataFrame 中
        predictions.loc[len(predictions)] = new_row
    except Exception as e:
        print(f"Error predicting for {crop} on {land_type}: {e}")

    # 保存预测结果到新的 CSV 文件
predictions.to_csv('crop_prices_predictions.csv', index=False)
```

农作物种植情况数据

```
    import pandas as pd

    file_crop_sales = './2023 年的农作物种植情况.xlsx'

    df_crop_sales = pd.read_excel(file_crop_sales)

    # 提取“销售单价/(元/斤)”中的信息并添加最低单价、最高单价、平均值三列
    # 这里我们假设“销售单价/(元/斤)”列中的数据是范围格式的，例如 "2.50-4.00"
```

```
    # 定义一个函数来提取最低值、最高值和平均值
    def extract_price_info(price_range):
        if isinstance(price_range, str) and '-' in price_range:
            low, high = map(float, price_range.split('-'))
            avg = (low + high) / 2
            return pd.Series([low, high, avg])
        else:
            return pd.Series([None, None, None])

    # 应用函数到“销售单价/(元/斤)”列并创建新的列
    df_crop_sales[['最低单价', '最高单价', '平均单价']] = df_crop_sales['销售单价/(元/斤)'].apply(extract_price_info)

    # 将处理后的数据保存到新的 Excel 文件中
    output_file_path_crop_sales = './处理后的农作物种植情况数据.xlsx'
    df_crop_sales.to_excel(output_file_path_crop_sales, index=False)

    output_file_path_crop_sales
```

粮食种类获取

```
    # 加载用户提供的两个 Excel 文件
    file_processed_crop_data = './处理后的农作物种植情况数据.xlsx'
    file_vegetable_data = './粮食种类 1.xlsx'

    # 读取 Excel 数据
    df_processed_crop_data = pd.read_excel(file_processed_crop_data, engine='openpyxl')
    df_vegetable_data = pd.read_excel(file_vegetable_data, engine='openpyxl')

    # 基于“作物编号”列将两个表合并，将第二个表中的“作物类型”列加到第一个表中
    # 左连接保留第一个表中的所有数据
    final_price_table = pd.merge(df_processed_crop_data, df_vegetable_data[['作物编号', '作物类型','是否是豆类']], on='作物编号', how='left')

    # 将合并后的数据保存到新的 Excel 文件中
    output_final_price_table = './23 年农作物价格表.xlsx'
    final_price_table.to_excel(output_final_price_table, index=False)

    output_final_price_table
```

价格表获取

```
    # 加载用户提供的两个 Excel 文件
    file_processed_crop_data = './处理后的农作物种植情况数据.xlsx'
    file_vegetable_data = './粮食种类 1.xlsx'

    # 读取 Excel 数据
    df_processed_crop_data = pd.read_excel(file_processed_crop_data, engine='op
enpyxl')
    df_vegetable_data = pd.read_excel(file_vegetable_data, engine='openpyxl')

    # 基于“作物编号”列将两个表合并，将第二个表中的“作物类型”列加到第一个表中
    # 左连接保留第一个表中的所有数据
    final_price_table = pd.merge(df_processed_crop_data, df_vegetable_data[['作
物编号', '作物类型','是否是豆类']], on='作物编号', how='left')

    # 将合并后的数据保存到新的 Excel 文件中
    output_final_price_table = './23 年农作物价格表.xlsx'
    final_price_table.to_excel(output_final_price_table, index=False)

    output_final_price_table
```

合并地块类型数据

```
    # 加载用户提供的更新后的两个 Excel 文件
    file_vegetable_updated = './2023 年各类蔬菜种植情况.xlsx'
    file_farm_land_updated = './乡村现有耕地.xlsx'

    # 读取蔬菜种植情况和乡村耕地信息的数据表
    df_vegetable_updated = pd.read_excel(file_vegetable_updated)
    df_farm_land_updated = pd.read_excel(file_farm_land_updated)

    # 尝试基于蔬菜种植地块和耕地地块信息合并
    # 假设蔬菜数据中的列为“种植地块”，耕地数据中的列为“利植地块”
    # 这里我们先检查两个表格中的列名以确保可以进行合并
    df_vegetable_updated.columns, df_farm_land_updated.columns

    # 使用“种植地块”和“地块名称”列进行合并操作，并保留地块类型信息
    # 基于蔬菜种植数据添加乡村耕地的地块类型信息

    # 合并蔬菜数据和耕地数据，左连接保留蔬菜表中的所有数据
    merged_df_updated = pd.merge(df_vegetable_updated, df_farm_land_updated, ho
```

```
w='left', left_on='种植地块', right_on='地块名称')


    # 将合并后的数据保存到一个新的 Excel 文件中
    output_file_path_updated = './合并后的蔬菜种植地块类型数据.xlsx'
    merged_df_updated.to_excel(output_file_path_updated, index=False)

    output_file_path_updated
```

蔬菜种植和利润数据

```
    # 加载用户提供的更新后的两个 Excel 文件
    file_vegetable_updated = './2023 年各类蔬菜种植情况.xlsx'
    file_farm_land_updated = './乡村现有耕地.xlsx'

    # 读取蔬菜种植情况和乡村耕地信息的数据表
    df_vegetable_updated = pd.read_excel(file_vegetable_updated)
    df_farm_land_updated = pd.read_excel(file_farm_land_updated)

    # 尝试基于蔬菜种植地块和耕地地块信息合并
    # 假设蔬菜数据中的列为“种植地块”，耕地数据中的列为“利植地块”
    # 这里我们先检查两个表格中的列名以确保可以进行合并
    df_vegetable_updated.columns, df_farm_land_updated.columns

    # 使用“种植地块”和“地块名称”列进行合并操作，并保留地块类型信息
    # 基于蔬菜种植数据添加乡村耕地的地块类型信息

    # 合并蔬菜数据和耕地数据，左连接保留蔬菜表中的所有数据
    merged_df_updated = pd.merge(df_vegetable_updated, df_farm_land_updated, ho
w='left', left_on='种植地块', right_on='地块名称')


    # 将合并后的数据保存到一个新的 Excel 文件中
    output_file_path_updated = './合并后的蔬菜种植地块类型数据.xlsx'
    merged_df_updated.to_excel(output_file_path_updated, index=False)

    output_file_path_updated
```

食用菌贪心处理

```
    import pandas as pd
    import matplotlib.pyplot as plt
```

```
    # 读取数据
    file_path = './2023 年农作物各类数据最终版 1.1.xlsx'
    data = pd.read_excel(file_path, sheet_name='Sheet1')

    # 按作物类型分组，并计算每组的平均每亩利润
    average_profit = data.groupby('作物类型')['每亩利润'].mean().reset_index()

    # 设置中文字体
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置字体为黑体
    plt.rcParams['axes.unicode_minus'] = False  # 正常显示负号

    # 绘制柱状图
    plt.figure(figsize=(10, 6))
    bars = plt.bar(average_profit['作物类型'], average_profit['每亩利润
'], width=0.5, color=['#FF9999','#66B3FF','#99FF99'])

    # 添加数据标签
    for bar in bars:
        yval = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2, yval + 500, round(yval, 2), h
a='center', va='bottom', fontsize=12)

    # 调整 X 轴和 Y 轴
    plt.ylim(0, max(average_profit['每亩利润']) * 1.1)  # 调整 Y 轴范围，使图像更紧
凑
    plt.xlabel('作物类型', fontsize=14)  # X 轴标签
    plt.ylabel('平均每亩利润', fontsize=14)  # Y 轴标签
    plt.title('不同作物类型的平均每亩利润', fontsize=16)  # 图表标题

    # 显示图像
    plt.tight_layout()
    plt.show()
```

Q1 地块数据处理

```
    import pandas as pd
    import numpy as np

    # 读取 2023 年数据
    file_path = './filled_crop_area_table.xlsx'  # 请根据需要调整路径
    data_2023 = pd.read_excel(file_path)

    # 清理列名，去除空白字符等问题
```

```
    data_2023.columns = data_2023.columns.str.strip()

    # 将所有作物列转换为数值类型，无法转换的值填充为 0
    data_2023.iloc[:, 2:] = data_2023.iloc[:, 2:].apply(pd.to_numeric, errors='
coerce').fillna(0)

    # 豆类作物列表，包括黄豆、黑豆、红豆、绿豆、爬豆
    legumes = ["黑豆", "黄豆", "红豆", "绿豆", "爬豆"]

    # 获取 2023 年的作物总面积
    crop_totals = data_2023.iloc[:, 2:].sum()

    # 获取需要操作的作物列的列名
    crop_columns = data_2023.columns[2:]  # 假设作物列从第 3 列开始

    # 基于 2023 年数据生成 2024 年数据
    previous_year_data = data_2023.copy()  # 使用 2023 年数据作为基础

    # 初始化 2024 年数据表格
    df_new = previous_year_data.copy()

    for i, row in df_new.iterrows():
        # 获取地块 2023 年的总面积限制
        total_area_2023 = data_2023.loc[i, crop_columns].sum()

        # 当前地块的已种植面积
        current_total_area = df_new.loc[i, crop_columns].sum()

        # 确保每块地种植的作物种类尽量少
        crops_to_plant = []
        # 避免连续两年种植相同作物
        # 确保数据全是数值类型并将字符串替换为 0
        row_data = pd.to_numeric(previous_year_data.loc[i, crop_columns], error
s='coerce').fillna(0)
        last_year_crops = row_data[row_data > 0].index.tolist()

        # 剔除前一年已经种过的作物，确保不种植 2023 年的作物
        available_crops = [crop for crop in crop_columns if crop not in last_ye
ar_crops]

        # 如果没有足够的作物可供选择，返回警告
        if not available_crops:
```

```
            print(f"Warning: No available crops for plot {i}, keeping previous
 year's crops.")

        # 选择新的作物
        if available_crops:  # 确保不种前一年相同的作物
            chosen_crop = np.random.choice(available_crops)
            area_to_add = crop_totals[chosen_crop] / len(df_new)
            if current_total_area + area_to_add <= total_area_2023:
                df_new.loc[i, chosen_crop] = area_to_add

        # 更新当前总种植面积
        current_total_area = df_new.loc[i, crop_columns].sum()

        # 如果当前面积没有变化或与 2023 年完全一致，则强制随机调整作物
        if (current_total_area == 0 or np.array_equal(previous_year_data.loc[i,
 crop_columns], df_new.loc[i, crop_columns])):
            # 选一个不同于 2023 年种植的作物
            if available_crops:
                new_crop = np.random.choice(available_crops)
                df_new.loc[i, new_crop] = crop_totals[new_crop] / len(df_new)

        # 更新当前总种植面积
        current_total_area = df_new.loc[i, crop_columns].sum()

        # 如果超出总面积限制，则调整比例
        if current_total_area > total_area_2023:
            scale_factor = total_area_2023 / current_total_area
            df_new.loc[i, crop_columns] = df_new.loc[i, crop_columns] * scale_f
actor

    # 保存生成的 2024 年 Excel 文件
    output_path = './2024 年 ABC 地块数据.xlsx'
    df_new.to_excel(output_path, index=False)

    output_path
```

粮食处理

```
    import pandas as pd
    import numpy as np

    # 初始化地块名称和面积
    plots_data = {
```

```
        '地块名称
': ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B
7', 'B8', 'B9', 'B10', 'B11', 'B12', 'B13', 'B14', 'C1', 'C2', 'C3', 'C4', 'C5
', 'C6'],
        '地块面积
': [80, 55, 35, 72, 68, 55, 60, 46, 40, 28, 25, 86, 55, 44, 50, 25, 60, 45, 35
, 20, 15, 13, 15, 18, 27, 20]
    }
    plots_df = pd.DataFrame(plots_data)

    # 定义作物及其对应的总种植面积
    crop_totals_new = {
        '黄豆': 147,
        '黑豆': 46,
        '红豆': 60,
        '绿豆': 96,
        '爬豆': 25,
        '小麦': 222,
        '玉米': 135,
        '谷子': 185,
        '高粱': 50,
        '黍子': 25,
        '荞麦': 15,
        '南瓜': 13,
        '红薯': 18,
        '莜麦': 35,
        '大麦': 20
    }

    # 定义每种作物的利润（从表格中提取的真实数据）
    crop_profits = {
        '黄豆': 950.00,
        '黑豆': 3241.67,
        '红豆': 2832.50,
        '绿豆': 2158.33,
        '爬豆': 2365.00,
        '小麦': 2416.67,
        '玉米': 2583.33,
        '谷子': 2390.00,
        '高粱': 2522.50,
        '黍子': 3473.33,
        '荞麦': 3816.67,
```

```
        '南瓜': 3275.00,
        '红薯': 4750.00,
        '莜麦': 1833.33,
        '大麦': 1441.67
    }

    # 三年内的种植规划
    num_years = 3
    num_plots = len(plots_df)
    num_crops = len(crop_totals_new)

    # 初始化利润矩阵 profit[i][y][k]
    profit = np.zeros((num_plots, num_years, num_crops))

    # 定义作物索引
    crop_list = list(crop_totals_new.keys())

    # 用于记录每块地的作物分配
    plot_crop_allocation = {plot: {} for plot in plots_df['地块名称']}

    # 初始化剩余种植面积（用于控制总面积限制）
    remaining_crop_totals = crop_totals_new.copy()

    # 第一年初始化种植方案，优先分配利润较大的作物，同时考虑剩余种植面积
    for index, row in plots_df.iterrows():
        plot_name = row['地块名称']
        plot_area = row['地块面积']
        remaining_plot_area = plot_area

        # 优先分配豆类作物，确保三年内至少种植一次
        beans = ['黄豆', '黑豆', '红豆', '绿豆', '爬豆']
        for crop in beans:
            if remaining_crop_totals[crop] > 0 and remaining_plot_area > 0:
                allocated_area = min(remaining_crop_totals[crop], remaining_plo
t_area)
                plot_crop_allocation[plot_name][crop] = allocated_area
                remaining_crop_totals[crop] -= allocated_area
                remaining_plot_area -= allocated_area

        # 然后分配其他作物
        for crop, profit_value in sorted(crop_profits.items(), key=lambda x: -x
[1]):
```

```
            if crop not in beans and remaining_crop_totals[crop] > 0 and remain
ing_plot_area > 0:
                allocated_area = min(remaining_crop_totals[crop], remaining_plo
t_area)
                plot_crop_allocation[plot_name][crop] = allocated_area
                remaining_crop_totals[crop] -= allocated_area
                remaining_plot_area -= allocated_area

    # 确保种植方案不会超过作物的总种植面积限制
    for crop, remaining_area in remaining_crop_totals.items():
        if remaining_area < 0:
            print(f"Warning: {crop} exceeded the total allowed area!")

    # 输出作物分配结果
    allocation_results = []

    for plot_name, crops in plot_crop_allocation.items():
        for crop, area in crops.items():
            allocation_results.append({'地块名称': plot_name, '作物名称
': crop, '种植面积': area})

    allocation_df = pd.DataFrame(allocation_results)

    # 展示分配结果
    print(allocation_df)

    # 保存分配结果为 Excel 文件
    allocation_df.to_excel("粮食作物分配结果_优化版.xlsx", index=False)
```

蔬菜处理

```
    import pandas as pd
    import numpy as np

    # 可用作物及其最大种植面积限制
    available_crops = [
        "刀豆", "包菜", "土豆", "大白菜", "小青菜", "油麦菜", "生菜", "白萝卜", "
空心菜", "红萝卜",
        "芸豆", "芹菜", "茄子", "菜花", "菠菜", "西红柿", "豇豆", "辣椒", "青椒
", "黄心菜", "黄瓜"
    ]

    crop_area_limits = {
```

```
        "刀豆": 13.2,
        "包菜": 0.9,
        "土豆": 15,
        "大白菜": 30,
        "小青菜": 10.9,
        "油麦菜": 0.9,
        "生菜": 0.6,
        "白萝卜": 25,
        "空心菜": 0.3,
        "红萝卜": 12,
        "芸豆": 1.8,
        "芹菜": 0.3,
        "茄子": 6.9,
        "菜花": 0.9,
        "菠菜": 0.3,
        "西红柿": 14.9,
        "豇豆": 11.8,
        "辣椒": 0.6,
        "青椒": 0.9,
        "黄心菜": 0.3,
        "黄瓜": 0.9
    }

    def generate_two_year_plan_with_correct_greenhouse_handling(crop_data_2023,
 crop_area_limits):
        """
        根据指定的地块和季节生成种植计划，生成 2024 和 2025 年的两年计划，处理普通大棚的第一季并跳过第二季，避免两年种植同样的作物。
        """
        crop_plan_2024 = pd.DataFrame(columns=['地块名称', '作物名称', '季节', '种植面积', '年份'])
        crop_plan_2025 = pd.DataFrame(columns=['地块名称', '作物名称', '季节', '种植面积', '年份'])

        # 记录每种作物的总种植面积
        total_area_planted_2024 = {crop: 0 for crop in crop_area_limits.keys()}
        total_area_planted_2025 = {crop: 0 for crop in crop_area_limits.keys()}

        # 第一季和第二季的地块
        season1_plots = ['D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8', 'E9', 'E10', 'E11', 'E12', 'E13', 'E14', 'E15', 'E16', 'F1', 'F2', 'F3', 'F4']
```

```
        season2_plots = ['D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'F1',
'F2', 'F3', 'F4']

        # Step 1: Generate the 2024 crop plan first
        def generate_year_plan(year, season_plots, total_area_planted, crop_pla
n, exclude_previous_year_plan=None):
            """
            Generate crop plan for a given year and season plots.
            """
            for season in [1, 2]:
                plots_to_use = season1_plots if season == 1 else season2_plots
                for plot in plots_to_use:
                    plot_data = crop_data_2023[crop_data_2023['种植地块
'] == plot]
                    if not plot_data.empty:
                        plot_area = plot_data['种植面积/亩'].values[0]
                        plot_type = plot_data['地块类型'].values[0]

                        # Assign crops for each season
                        if plot_type == '水浇地':
                            if season == 1:
                                available_crops_season = [crop for crop in avai
lable_crops if crop not in ['大白菜', '白萝卜', '红萝卜']]
                            else:
                                available_crops_season = ['大白菜', '白萝卜', '
红萝卜']
                        elif plot_type == '普通大棚':
                            if season == 1:
                                # 处理普通大棚的第一季，排除大白菜、白萝卜、红萝卜
                                available_crops_season = [crop for crop in avai
lable_crops if crop not in ['大白菜', '白萝卜', '红萝卜']]
                            else:
                                # 跳过普通大棚的第二季
                                continue
                        elif plot_type == '智慧大棚':
                            available_crops_season = [crop for crop in availabl
e_crops if crop not in ['大白菜', '白萝卜', '红萝卜']]
                        else:
                            available_crops_season = []

                        # For 2025, exclude the crop used in 2024 to avoid rep
etition
```

```
                        if exclude_previous_year_plan is not None:
                            previous_year_crop = exclude_previous_year_plan.get
(plot, {}).get(season, None)
                            if previous_year_crop:
                                available_crops_season = [crop for crop in avai
lable_crops_season if crop != previous_year_crop]

                        # Select a crop for the plot
                        selected_crop = None
                        for crop in available_crops_season:
                            if total_area_planted[crop] < crop_area_limits[crop
]:
                                selected_crop = crop
                                break

                        if selected_crop is None:
                            continue

                        # Calculate planting area and update the total planted
 area
                        remaining_area_for_crop = crop_area_limits[selected_cro
p] - total_area_planted[selected_crop]
                        planting_area = min(plot_area, remaining_area_for_crop)

                        total_area_planted[selected_crop] += planting_area

                        # Add to the crop plan
                        new_row = pd.DataFrame({
                            '地块名称': [plot],
                            '作物名称': [selected_crop],
                            '季节': [season],
                            '种植面积': [planting_area],
                            '年份': [year]
                        })
                        crop_plan = pd.concat([crop_plan, new_row], ignore_inde
x=True)

                        # Track the previous year crops for 2024 (to avoid rep
etition in 2025)
                        if exclude_previous_year_plan is not None:
                            exclude_previous_year_plan[plot][season] = selected
_crop
```

```
            return crop_plan

        # Step 2: Create the 2024 plan
        exclude_previous_year_plan_2024 = {plot: {} for plot in season1_plots +
 season2_plots}
        crop_plan_2024 = generate_year_plan(2024, season1_plots + season2_plots
, total_area_planted_2024, crop_plan_2024)

        # Step 3: Create the 2025 plan, avoiding repeating 2024 crops
        crop_plan_2025 = generate_year_plan(2025, season1_plots + season2_plots
, total_area_planted_2025, crop_plan_2025, exclude_previous_year_plan_2024)

        # Combine both year plans
        full_plan = pd.concat([crop_plan_2024, crop_plan_2025], ignore_index=Tr
ue)

        return full_plan

    # Assuming crop_data_2023 is loaded from an external source

    # Example:
    # crop_data_2023 = pd.read_excel('path_to_crop_data.xlsx')

    # 生成 2024 和 2025 年的种植计划
    corrected_two_year_plan = generate_two_year_plan_with_correct_greenhouse_ha
ndling(crop_data_2023, crop_area_limits)

    # 保存生成的种植计划为 Excel 文件
    output_path_corrected = './Final_2024_2025_Crop_Plan_Corrected_Greenhouse_H
andled.xlsx'
    corrected_two_year_plan.to_excel(output_path_corrected, index=False)

    print(f"种植计划已保存至: {output_path_corrected}")
```

利润求解

```
    import pandas as pd

    # 加载 Excel 文件
    file1_path = './2023 年农作物各类数据最终版 1.1.xlsx'
    file2_path = './result1_1_0907_1.30 点.xlsx'
```

```
    # 读取两个文件中的相关表格
    file1_data = pd.read_excel(file1_path, sheet_name='Sheet1')
    file2_data_2026 = pd.read_excel(file2_path, sheet_name='2026')

    # 从第一个表格提取作物名称和每亩利润，并将作物名称作为索引
    profit_per_acre = file1_data[['作物名称', '每亩利润']].set_index('作物名称')

    # 删除重复的作物名称，保留第一个出现的利润值
    profit_per_acre_cleaned = profit_per_acre[~profit_per_acre.index.duplicated
(keep='first')]

    # 获取 2026 年表格中的作物列（从第三列开始）
    crop_columns = file2_data_2026.columns[2:]

    # 确保所有种植面积列的数据类型为数值，处理非数值数据
    file2_data_2026[crop_columns] = file2_data_2026[crop_columns].apply(pd.to_n
umeric, errors='coerce').fillna(0)

    # 初始化一个 DataFrame 来存储利润计算结果
    profits = pd.DataFrame(index=file2_data_2026.index)

    # 计算每个作物的利润（种植面积 * 每亩利润）
    for crop in crop_columns:
        if crop in profit_per_acre_cleaned.index:
            profits[crop] = file2_data_2026[crop] * profit_per_acre_cleaned.loc
[crop, '每亩利润']
        else:
            profits[crop] = 0  # 如果作物没有在利润表中出现，则设为 0 利润

    # 计算每个地块的总利润
    file2_data_2026['总利润'] = profits.sum(axis=1)

    # 计算 2026 年的总利润
    total_profit_2026 = file2_data_2026['总利润'].sum()

    # 输出 2026 年的总利润
    print("2026 年的总利润为: ", total_profit_2026)
```

改进遗传算法求解

```
    import numpy as np
    import pandas as pd
```

```
    import random
    import matplotlib.pyplot as plt

    # 加载 Excel 文件中的数据
    file1_path = './result1_1_0907_1.30 点.xlsx'
    file2_path = './2023 年农作物各类数据最终版 1.1.xlsx'

    # 加载七个年份的数据（2024 年至 2030 年）
    years = ['2024', '2025', '2026', '2027', '2028', '2029', '2030']
    area_data = {year: pd.read_excel(file1_path, sheet_name=year) for year in y
ears}

    # 加载第二个文件的种植利润和成本等数据
    file2_data = pd.read_excel(file2_path, sheet_name='Sheet1')

    # 确保相关列的数据类型为数值格式，并清理列名
    for year in area_data:
        area_data[year] = area_data[year].apply(pd.to_numeric, errors='coerce')
.fillna(0)

    file2_data = file2_data.apply(pd.to_numeric, errors='coerce', axis=1).filln
a(0)
    file2_data.columns = file2_data.columns.str.strip()  # 去除列名中的空格

    # 设置中文字体
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置字体为黑体
    plt.rcParams['axes.unicode_minus'] = False  # 正常显示负号

    # 从 file2_data 中提取豆类作物的编号
    bean_crops = file2_data[file2_data['是否是豆类'] == 1]['作物编号'].unique()

    # 提取作物名称，使用实际作物名称
    crop_names = area_data['2024'].columns[2:]  # 从第三列开始是作物名称

    # 作物利润和价格
    yield_per_unit = file2_data['每亩利润'].values  # 每亩利润
    price_per_unit = file2_data['加权单价/(元/斤)'].values  # 销售价格
    planting_cost = file2_data['每亩利润'].values  # 种植成本
    expected_sales = file2_data['预售量/亩'].values  # 预售量

    # 打印调试信息，确保数据正确
    print("Yield per unit:", yield_per_unit)
```

```
    print("Price per unit:", price_per_unit)
    print("file2_data:", file2_data.head())

    # 遗传算法参数
    POPULATION_SIZE = 50   # 种群规模
    num_crops = len(crop_names)  # 匹配到的作物数量
    CHROMOSOME_LENGTH = num_crops * 7  # 每个染色体表示 7 年的作物种植面积
    MUTATION_RATE = 0.05    # 变异概率
    CROSSOVER_RATE = 0.8    # 交叉概率
    NUM_GENERATIONS = 100   # 迭代次数

    # 重茬惩罚系数
    REPLANT_PENALTY = 0.8  # 如果同一块地连续种植同一作物，利润降低 20%

    # 适应度函数：根据七年种植面积计算总收益
    def fitness_function(population):
        fitness_scores = []

        for chromosome in population:
            total_profit = 0
            bean_crop_planted = np.zeros(num_crops)  # 记录豆类作物的种植年份

            # 按年份计算
            for year_idx, year in enumerate(years):  # 7 年循环
                for i in range(num_crops):  # 遍历每个作物

                    # 获取当前年的种植面积
                    index = year_idx * num_crops + i
                    area = chromosome[index]  # 获取当前年的种植面积

                    # 检查重茬种植
                    if year_idx > 0 and chromosome[(year_idx-1) * num_crops + i
] > 0:
                        total_profit += REPLANT_PENALTY * yield_per_unit[i] * a
rea * price_per_unit[i]
                    else:
                        total_profit += yield_per_unit[i] * area * price_per_un
it[i]

                    # 豆类作物轮种检查
                    if file2_data.iloc[i]['是否是豆类'] == 1:  # 如果当前作物是豆
类
```

```
                        bean_crop_planted[i] += 1
                    elif bean_crop_planted[i] < 3:  # 未满足三年种植一次豆类作物
                        total_profit -= 500  # 惩罚措施

            fitness_scores.append(total_profit)

        return np.array(fitness_scores)

    # 初始化种群：每个个体表示 7 年的种植面积方案
    def initialize_population():
        population = []
        for _ in range(POPULATION_SIZE):
            chromosome = np.random.uniform(0, 100, CHROMOSOME_LENGTH)  # 随机生成 0 到 100 亩的种植面积
            population.append(chromosome)
        return np.array(population)

    # 选择操作：轮盘赌选择
    def selection(population, fitness_scores):
        total_fitness = np.sum(fitness_scores)
        selection_probs = fitness_scores / total_fitness
        selected_indices = np.random.choice(range(POPULATION_SIZE), size=POPULATION_SIZE, p=selection_probs)
        return population[selected_indices]

    # 交叉操作：单点交叉
    def crossover(parent1, parent2):
        if random.random() < CROSSOVER_RATE:
            crossover_point = random.randint(1, CHROMOSOME_LENGTH - 1)
            child1 = np.concatenate((parent1[:crossover_point], parent2[crossover_point:]))
            child2 = np.concatenate((parent2[:crossover_point], parent1[crossover_point:]))
            return child1, child2
        else:
            return parent1, parent2

    # 变异操作
    def mutate(chromosome):
        for i in range(CHROMOSOME_LENGTH):
            if random.random() < MUTATION_RATE:
                chromosome[i] = np.random.uniform(0, 100)  # 随机改变某个地块的
```

```
种植面积
        return chromosome

    # 遗传算法主循环，并可视化输出
    def genetic_algorithm_with_visualization():
        population = initialize_population()  # 初始化种群
        best_fitness_history = []  # 记录每代的最佳适应度

        for generation in range(NUM_GENERATIONS):
            fitness_scores = fitness_function(population)  # 计算适应度
            population = selection(population, fitness_scores)  # 选择操作

            # 生成下一代
            next_generation = []
            for i in range(0, POPULATION_SIZE, 2):
                parent1, parent2 = population[i], population[i+1]
                child1, child2 = crossover(parent1, parent2)
                next_generation.append(mutate(child1))
                next_generation.append(mutate(child2))

            population = np.array(next_generation)

            # 输出每一代的最佳适应度
            best_fitness = np.max(fitness_scores)
            best_fitness_history.append(best_fitness)

        # 最终结果
        final_fitness_scores = fitness_function(population)
        best_fitness = np.max(final_fitness_scores)
        best_individual = population[np.argmax(final_fitness_scores)]
        print(f"Final solution: Best fitness = {best_fitness}")

        # 返回每代的最佳适应度记录
        return best_fitness_history, best_individual

    # 运行算法并获取结果
    best_fitness_history, best_individual = genetic_algorithm_with_visualizatio
n()

    # 可视化展示每代的最佳适应度
    plt.figure(figsize=(10, 5))
    plt.plot(range(1, NUM_GENERATIONS + 1), best_fitness_history, marker='o', l
```

```
inestyle='-', color='b', label='每代最佳适应度')
    plt.title('每代最佳适应度变化趋势')
    plt.xlabel('迭代次数（代数）')
    plt.ylabel('适应度值（收益）')
    plt.grid(True)
    plt.legend()
    plt.show()
```

模拟退火求解

```
    import numpy as np
    import pandas as pd
    import random
    import matplotlib.pyplot as plt

    # 加载 Excel 文件中的数据
    file1_path = './result1_1_0907_1.30 点.xlsx'
    file2_path = './2023 年农作物各类数据最终版 1.1.xlsx'
    file3_path = './乡村现有耕地.xlsx'

    # 加载七个年份的数据（2024 年至 2030 年）
    years = ['2024', '2025', '2026', '2027', '2028', '2029', '2030']
    area_data = {year: pd.read_excel(file1_path, sheet_name=year) for year in y
ears}

    # 加载第二个文件的种植利润和成本等数据
    file2_data = pd.read_excel(file2_path, sheet_name='Sheet1')

    # 加载第三个文件的地块面积限制数据
    land_limits = pd.read_excel(file3_path, sheet_name='Sheet1')

    # 确保相关列的数据类型为数值格式，并清理列名
    for year in area_data:
        area_data[year] = area_data[year].apply(pd.to_numeric, errors='coerce')
.fillna(0)

    file2_data = file2_data.apply(pd.to_numeric, errors='coerce', axis=1).filln
a(0)
    file2_data.columns = file2_data.columns.str.strip()  # 去除列名中的空格

    land_limits.columns = land_limits.columns.str.strip()  # 去除列名中的空格
    land_limits_dict = dict(zip(land_limits['地块名称'], land_limits['地块面积/
亩']))  # 创建地块面积限制的字典
```

```
    # 设置中文字体
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置字体为黑体
    plt.rcParams['axes.unicode_minus'] = False  # 正常显示负号

    # 从 file2_data 中提取豆类作物的编号
    bean_crops = file2_data[file2_data['是否是豆类'] == 1]['作物编号'].unique()

    # 提取作物名称，使用实际作物名称
    crop_names = area_data['2024'].columns[2:]  # 从第三列开始是作物名称

    # 模拟退火算法参数
    INITIAL_TEMPERATURE = 1000   # 初始温度
    FINAL_TEMPERATURE = 1  # 终止温度
    COOLING_RATE = 0.95  # 温度下降速率
    MAX_ITERATIONS = 1000  # 最大迭代次数

    # 重茬惩罚系数
    REPLANT_PENALTY = 0.8  # 如果同一块地连续种植同一作物，利润降低 20%

    # 获取作物利润，首先尝试根据作物名和地块名匹配，如果失败则仅根据作物名匹配
    def get_crop_profit(crop_name, plot_name):
        # 尝试通过作物名和地块名同时匹配
        match = file2_data[(file2_data['作物名称
'] == crop_name) & (file2_data['种植地块'] == plot_name)]
        if not match.empty:
            return match['每亩利润'].values[0]

        # 如果没有找到同时匹配的项，则只根据作物名匹配
        match_by_crop = file2_data[file2_data['作物名称'] == crop_name]
        if not match_by_crop.empty:
            return match_by_crop['每亩利润'].values[0]

        # 如果都未匹配到，返回 0 表示没有利润数据
        return 0

    # 计算目标函数：根据七年种植面积计算总收益
    def fitness_function(solution):
        total_profit = 0
        num_crops = len(crop_names)
        num_plots = len(area_data['2024']['地块名'])  # 假设每年地块数量相同
        bean_crop_planted = np.zeros(num_crops)  # 记录豆类作物的种植年份
```

```
        for year_idx, year in enumerate(years):  # 7 年循环
            for plot_idx in range(num_plots):  # 遍历每个地块
                for i, crop_name in enumerate(crop_names):  # 遍历每个作物
                    plot_name = area_data[year]['地块名'].iloc[plot_idx]  # 获
取当前的地块名
                    area = solution[year_idx * num_crops * num_plots + i * num_
plots + plot_idx]  # 获取该地块的种植面积

                    # 获取对应的利润
                    crop_profit_per_acre = get_crop_profit(crop_name, plot_name
)

                    # 计算该地块的总利润（面积 * 每亩利润）
                    total_profit += crop_profit_per_acre * area

                    # 检查重茬种植
                    if year_idx > 0:
                        prev_year_area = solution[(year_idx - 1) * num_crops *
num_plots + i * num_plots + plot_idx]
                        if prev_year_area > 0 and area > 0:  # 连续种植
                            total_profit *= REPLANT_PENALTY  # 重茬作物的收益减
少 20%

                    # 豆类作物轮种检查
                    if i in bean_crops:  # 如果当前作物是豆类
                        bean_crop_planted[i] += 1
                    elif bean_crop_planted[i] < 3:  # 未满足三年种植一次豆类作物
                        total_profit -= 500  # 惩罚措施

        return total_profit

    # 初始化解
    def initialize_solution():
        num_crops = len(crop_names)
        num_plots = len(area_data['2024']['地块名'])
        solution = np.random.uniform(0, 100, num_crops * num_plots * len(years)
)  # 随机生成种植面积
        return solution

    # 生成新解（扰动当前解）
    def perturb_solution(solution):
```

```
        new_solution = solution.copy()
        idx = np.random.randint(0, len(solution))
        new_solution[idx] = np.random.uniform(0, 100)  # 对某个地块的种植面积进行
随机扰动
        return new_solution

    # 模拟退火算法主循环
    def simulated_annealing():
        current_solution = initialize_solution()
        current_fitness = fitness_function(current_solution)
        best_solution = current_solution.copy()
        best_fitness = current_fitness

        fitness_history = []  # 记录每代的适应度变化
        temperature = INITIAL_TEMPERATURE

        for iteration in range(MAX_ITERATIONS):
            new_solution = perturb_solution(current_solution)
            new_fitness = fitness_function(new_solution)

            if new_fitness > current_fitness:
                current_solution = new_solution
                current_fitness = new_fitness
            else:
                acceptance_probability = np.exp((new_fitness - current_fitness)
 / temperature)
                if random.random() < acceptance_probability:
                    current_solution = new_solution
                    current_fitness = new_fitness

            if current_fitness > best_fitness:
                best_solution = current_solution
                best_fitness = current_fitness

            fitness_history.append(best_fitness)  # 保存每次迭代的最佳适应度

            temperature *= COOLING_RATE

            if temperature < FINAL_TEMPERATURE:
                break

            print(f"Iteration {iteration}: Best fitness = {best_fitness}, Temp
```

```
erature = {temperature}")

        return best_solution, best_fitness, fitness_history

    # 运行模拟退火算法并获取最优解
    best_solution, best_fitness, fitness_history = simulated_annealing()
    print(f"Best solution found: Fitness = {best_fitness}")

    # 可视化适应度的变化趋势
    plt.figure(figsize=(10, 5))
    plt.plot(fitness_history, marker='o', linestyle='-', color='b', label='最佳
适应度')
    plt.title('模拟退火过程中的适应度变化')
    plt.xlabel('迭代次数')
    plt.ylabel('适应度值（收益）')
    plt.grid(True)
    plt.legend()
    plt.show()
```