**第一题**

**定义对象以存储表格数据**

class Field:

    def \_\_init\_\_(self, field\_name, field\_type, field\_area, crop\_and\_season, season, seasonmonth, planted\_crop):

        self.field\_name = field\_name

        self.field\_type = field\_type

        self.field\_area = field\_area

        self.crop\_and\_season = crop\_and\_season

        self.season = season

        self.seasonmonth = seasonmonth

        self.planted\_crop = planted\_crop

    def \_\_str\_\_(self):

        return f'Field(planted\_crop={self.planted\_crop}, field\_name={self.field\_name}, field\_type={self.field\_type}, field\_area={self.field\_area}, crop\_and\_season={self.crop\_and\_season}, season={self.season}, seasonmonth={self.seasonmonth})'

class Crop:

    def \_\_init\_\_(self, crop\_id, crop\_name, crop\_type, planting\_fields, crop\_price):

        self.crop\_id = crop\_id

        self.crop\_name = crop\_name

        self.crop\_type = crop\_type

        self.planting\_fields = planting\_fields

        self.crop\_price = crop\_price

    def \_\_repr\_\_(self):

        return f"Crop(crop\_id={self.crop\_id}, crop\_name={self.crop\_name}, crop\_type={self.crop\_type}, planting\_fields={self.planting\_fields}, crop\_price={self.crop\_price})"

**初步构建优化模型**

def create\_variables(new\_fields, crops):

    # 创建决策变量

    variables = {}

    binary\_variables = {}

    for field in new\_fields:

        for crop in crops:

            var\_name = f"{field.field\_name}\_{crop.crop\_name}\_{field.season}"

            variables[(field.field\_name, crop.crop\_name, field.season)] = pulp.LpVariable(var\_name, lowBound=0)

            binary\_var\_name = f"binary\_{var\_name}"

            binary\_variables[(field.field\_name, crop.crop\_name, field.season)] = pulp.LpVariable(binary\_var\_name, cat='Binary')

    # 创建实际销售量变量

    actual\_sales = {}

    for crop in crops:

        actual\_sales[crop.crop\_name] = pulp.LpVariable(f"actual\_sales\_{crop.crop\_name}", lowBound=0)

    # 创建超额部分变量

    excess\_yield = {}

    for crop in crops:

        excess\_yield[crop.crop\_name] = pulp.LpVariable(f"excess\_yield\_{crop.crop\_name}", lowBound=0)

    return variables, binary\_variables, actual\_sales, excess\_yield

def define\_objective\_function(variables, actual\_sales, excess\_yield, crops, new\_fields, k):

    # 收入部分：实际销售量 \* 作物价格

    revenue = pulp.lpSum([

        actual\_sales[crop.crop\_name] \* crop.crop\_price

        for crop in crops

    ])

    # 成本部分：种植成本

    cost = pulp.lpSum([

        variables[(field.field\_name, crop.crop\_name, field.season)] \* get\_cost(field, crop)

        for field in new\_fields for crop in crops

    ])

    # 超额部分处理：超额部分 \* 作物价格 \* 系数

    excess\_handling = pulp.lpSum([

        excess\_yield[crop.crop\_name] \* crop.crop\_price \* k

        for crop in crops

    ])

    # 总利润 = 收入 - 成本 + 超额部分处理

    profit = revenue - cost + excess\_handling

    return profit

def add\_constraints(prob, variables, binary\_variables, actual\_sales, excess\_yield,

                    crops, new\_fields, expected\_sales\_data, df\_last1\_result, df\_last2\_result,

                    min\_area\_percent, max\_plots):

    # 添加约束条件

    # 地块面积限制

    for field in new\_fields:

        prob += pulp.lpSum([variables[(field.field\_name, crop.crop\_name, field.season)] for crop in crops]) <= field.field\_area

    # 作物种植限制

    for field in new\_fields:

        for crop in crops:

            if not check\_crop\_constraints(field, crop):

                prob += variables[(field.field\_name, crop.crop\_name, field.season)] == 0

    # 重茬限制

    for field in new\_fields:

        for crop in crops:

            if not check\_rotation\_constraints(field, crop, df\_last1\_result):

                prob += variables[(field.field\_name, crop.crop\_name, field.season)] == 0

    # 豆类作物种植要求

    for field in new\_fields:

        # 约束：确保三年内至少有一年种植过豆类作物

        prob += (

            pulp.lpSum([

                binary\_variables[(field.field\_name, crop.crop\_name, field.season)]

                for crop in crops if crop.crop\_type in ['粮食（豆类）', '蔬菜（豆类）']

            ]) +

            (df\_last1\_result[(df\_last1\_result['种植地块'] == field.field\_name) &

                            (df\_last1\_result['种植季次'] == field.season)]

            .iloc[:, 2:]

            .apply(lambda row: any(row[crop\_name] > 0 for crop\_name in row.index if crop\_name in [crop.crop\_name for crop in crops if crop.crop\_type in ['粮食（豆类）', '蔬菜（豆类）']]), axis=1)

            .any()) +

            (df\_last2\_result[(df\_last2\_result['种植地块'] == field.field\_name) &

                            (df\_last2\_result['种植季次'] == field.season)]

            .iloc[:, 2:]

            .apply(lambda row: any(row[crop\_name] > 0 for crop\_name in row.index if crop\_name in [crop.crop\_name for crop in crops if crop.crop\_type in ['粮食（豆类）', '蔬菜（豆类）']]), axis=1)

            .any()) >= 1

        )

    # 实际销售量约束

    for crop in crops:

        actual\_yield = pulp.lpSum([

            variables[(field.field\_name, crop.crop\_name, field.season)] \* get\_yield(field, crop)

            for field in new\_fields

        ])

        expected\_sales = get\_expected\_sales(crop, expected\_sales\_data)

        prob += actual\_sales[crop.crop\_name] <= actual\_yield

        prob += actual\_sales[crop.crop\_name] <= expected\_sales

        # 超额部分约束

        prob += excess\_yield[crop.crop\_name] == actual\_yield - expected\_sales # 这里不是>=，应该是==？！

        prob += excess\_yield[crop.crop\_name] >= 0

    # 作物在单个地块某一季节种植面积不宜太小

    for field in new\_fields:

        for crop in crops:

            var = variables[(field.field\_name, crop.crop\_name, field.season)]

            binary\_var = binary\_variables[(field.field\_name, crop.crop\_name, field.season)]

            min\_area = min\_area\_percent \* field.field\_area

            prob += var >= min\_area \* binary\_var

            prob += var <= field.field\_area \* binary\_var

    # 作物种植不宜太分散

    for crop in crops:

        for field\_type in set(field.field\_type for field in new\_fields):

            if field\_type not in ['普通大棚', '智慧大棚']:

                prob += pulp.lpSum([

                    binary\_variables[(field.field\_name, crop.crop\_name, field.season)]

                    for field in new\_fields if field.field\_type == field\_type

                ]) <= max\_plots

return prob

**第二题**

**更新数据**

def update\_expected\_sales\_data(expected\_sales\_data, expected\_sales\_data\_2023, std\_devs):

    # 定义增长率的均值

    wheat\_growth\_mean = (0.05 + 0.1) / 2

    corn\_growth\_mean = (0.05 + 0.1) / 2

    others\_growth\_mean = (-0.05 + 0.05) / 2

    # 遍历 expected\_sales 中的每一行

    for index, row in expected\_sales\_data.iterrows():

        crop\_name = row['作物名称']

        if crop\_name == '小麦':

            growth\_rate = np.random.normal(loc=wheat\_growth\_mean, scale=std\_devs[0])

            expected\_sales\_data.at[index, '预期销售量/斤'] \*= (1 + growth\_rate)

        elif crop\_name == '玉米':

            growth\_rate = np.random.normal(loc=corn\_growth\_mean, scale=std\_devs[1])

            expected\_sales\_data.at[index, '预期销售量/斤'] \*= (1 + growth\_rate)

        else:

            # 从 expected\_sales\_2023 中找到对应的行

            row\_2023 = expected\_sales\_data\_2023[expected\_sales\_data\_2023['作物名称'] == crop\_name]

            if not row\_2023.empty:

                base\_sales = row\_2023.iloc[0]['预期销售量/斤']

                growth\_rate = np.random.normal(loc=others\_growth\_mean, scale=std\_devs[2])

                expected\_sales\_data.at[index, '预期销售量/斤'] = base\_sales \* (1 + growth\_rate)

    return expected\_sales\_data

def update\_crops(crops, std\_devs):

    # 定义增长率的均值

    yield\_growth\_mean = (-0.1 + 0.1) / 2

    cost\_growth\_mean = 0.05

    price\_growth\_mean\_vegetables = 0.05

    price\_growth\_mean\_mushrooms = (0.01 + 0.05) / 2

    # 遍历 crops 列表

    for crop in crops:

        for field in crop.planting\_fields:

            # 更新亩产量

            yield\_growth\_rate = np.random.normal(loc=yield\_growth\_mean, scale=std\_devs[3])

            field[2] \*= (1 + yield\_growth\_rate)

            # 更新种植成本

            cost\_growth\_rate = np.random.normal(loc=cost\_growth\_mean, scale=std\_devs[4])

            field[3] \*= (1 + cost\_growth\_rate)

        # 更新作物价格

        if crop.crop\_type in ['粮食', '粮食（豆类）', '粮食（水稻）']:

            price\_growth\_rate = np.random.normal(loc=0, scale=std\_devs[5])

            crop.crop\_price \*= (1 + price\_growth\_rate)

        elif crop.crop\_type in ['蔬菜', '蔬菜（水二）', '蔬菜（豆类）']:

            price\_growth\_rate = np.random.normal(loc=price\_growth\_mean\_vegetables, scale=std\_devs[6])

            crop.crop\_price \*= (1 + price\_growth\_rate)

        elif crop.crop\_type == '食用菌':

            if crop.crop\_name == '羊肚菌':

                crop.crop\_price \*= 0.95  # 下降5%

            else:

                price\_growth\_rate = np.random.normal(loc=price\_growth\_mean\_mushrooms, scale=std\_devs[7])

                crop.crop\_price \*= (1 + price\_growth\_rate)

    return crops

**多目标优化计算分数**

def calculate\_stability(past\_profits):

    # 对往年利润进行一阶差分

    profit\_diffs = np.diff(past\_profits)

    # 计算标准差

    stability = np.std(profit\_diffs)

    return stability

def calculate\_score(profits, stability, weights):

    score = profits \* weights[0] - stability \* weights[1]

    return score

objective\_value = optimize\_planting\_strategy(new\_fields, crops, expected\_sales\_data, df\_last1\_result, df\_last2\_result, list)

df\_template\_list.append(update\_template(df\_template, new\_fields))

past\_profits.append(objective\_value)

profits[year - 2023] = objective\_value

stability = calculate\_stability(past\_profits)

stabilitys[year - 2023] = stability

score\_now = calculate\_score(objective\_value, stability, weights)

**第三题**

**添加约束条件**

# 引入相关性约束。传入总体的相关性矩阵

for i, crop1 in enumerate(crops):

    for j, crop2 in enumerate(crops):

        if i != j:

            prob += actual\_sales[crop1.crop\_name] \* correlation\_matrix[i, j] <= actual\_sales[crop2.crop\_name]

# 引入回归关系

for crop in crops:

    price = crop.crop\_price

    cost = get\_cost(crop)

    # 根据作物类型选择对应的回归参数

    params = regression\_params\_dict.get(crop.crop\_type)

    if params:

        prob += actual\_sales[crop.crop\_name] == params['intercept'] + params['price\_coef'] \* price + params['cost\_coef'] \* cost

    prob += actual\_sales[crop.crop\_name] == params['intercept'] + params['price\_coef'] \* price + params['cost\_coef'] \* cost

return prob