C 题 蔬菜类商品的自动定价与补货决策

在生鲜商超中，一般蔬菜类商品的保鲜期都比较短，且品相随销售时间的增加而变差， 大部分品种如当日未售出，隔日就无法再售。因此，商超通常会根据各商品的历史销售和需 求情况每天进行补货。 由于商超销售的蔬菜品种众多、产地不尽相同，而蔬菜的进货交易时间通常在凌晨 3:00- 4:00，为此商家须在不确切知道具体单品和进货价格的情况下，做出当日各蔬菜品类的补货 决策。蔬菜的定价一般采用“成本加成定价”方法，商超对运损和品相变差的商品通常进行 打折销售。可靠的市场需求分析，对补货决策和定价决策尤为重要。从需求侧来看，蔬菜类 商品的销售量与时间往往存在一定的关联关系；从供给侧来看，蔬菜的供应品种在 4 月至 10 月较为丰富，商超销售空间的限制使得合理的销售组合变得极为重要。 附件 1 给出了某商超经销的 6 个蔬菜品类的商品信息；附件 2 和附件 3 分别给出了该 商超 2020 年 7 月 1 日至 2023 年 6 月 30 日各商品的销售流水明细与批发价格的相关数据； 附件 4 给出了各商品近期的损耗率数据。请根据附件和实际情况建立数学模型解决以下问题:

问题 1 蔬菜类商品不同品类或不同单品之间可能存在一定的关联关系，请分析蔬菜各 品类及单品销售量的分布规律及相互关系。

思路

1. 数据预处理：

- 首先，从附件 2 中提供的销售流水明细数据中，筛选出与蔬菜类商品相关的数据。
- 对数据进行清洗，包括去除缺失值、异常值，以确保数据的质量。
- 将销售日期与销售数量按蔬菜品类或单品进行分类。

2. 描述性统计分析：

- 对每个蔬菜品类和单品的销售量进行统计分析，计算平均值、方差、中位数等统计量。
- 绘制销售量的直方图和箱线图，以了解销售量的分布情况。

3. 相关性分析：

- 利用统计工具（如相关系数）分析不同蔬菜品类或单品之间的销售量关联性。
- 绘制散点图来可视化销售量之间的关系。
- **数据准备**：根据问题 1 的目标，选择需要分析的蔬菜品类或单品的销售数据，将其提取出来。

- **计算相关性**：使用统计方法（如皮尔逊相关系数、斯皮尔曼相关系数）来计算不同蔬菜品类或单品之间的销售量相关性。
- **可视化**：绘制散点图或热力图，将相关性可视化，以便更清晰地观察不同品类或单品之间的关系。

4. 时间序列分析：

- 对于每个蔬菜品类或单品，可以将销售量随时间的变化进行时间序列分析，识别销售趋势和季节性。
- 使用时间序列模型（例如 ARIMA 模型）来预测未来的销售量。
- **数据准备**：选取需要分析的蔬菜品类或单品的销售数据，并按时间进行排序。
- **时间序列分解**：将时间序列数据拆解为趋势、季节性和噪声成分，以便更好地理解销售趋势。
- **模型拟合**：根据时间序列的性质，选择合适的时间序列模型，如 ARIMA 模型。拟合模型并进行参数估计。
- **预测未来销售量**：使用拟合的模型进行未来销售量的预测，以了解可能的销售趋势。

5. 聚类分析：

- 使用聚类分析方法（如 K 均值聚类）来将蔬菜品类或单品进行分组，找出销售行为相似的商品群组。
- 进一步研究每个群组内的销售量关系。
- **数据准备**：选择要进行聚类分析的蔬菜品类或单品的销售数据，确保数据包含足够多的特征。
- **特征选择**：如果有多个特征（例如销售数量、销售额、毛利率等），选择合适的特征来进行聚类。

- **标准化**：对特征进行标准化，以确保它们在相同的尺度上。
- **聚类算法**：选择适当的聚类算法，如 K 均值聚类。确定要分成的聚类数目。
- **聚类结果可视化**：绘制聚类结果的散点图或簇中心的可视化图，以便观察不同蔬菜品类或单品的聚类情况。

6. 可视化分析：

- 利用数据可视化工具，绘制热力图、相关性图、时间序列图等，以更清晰地展示销售量分布规律和关系。

7. 结果解释与应用：

- 分析得到的结果可以帮助商超了解不同蔬菜品类和单品之间的销售量趋势、关联关系和季节性。
- 这些信息可以用于制定补货策略，例如在销售量较大的蔬菜品类之间进行替代补货，以减少库存损失。
- 还可以用于定价策略，例如根据销售量关系来调整不同品类或单品的定价，提高销售收益。

1. 数据准备：

- 使用附件 2 中的数据，提取各蔬菜品类的销售数量、批发价格和损耗率。
- 对数据进行清洗和处理，确保数据的准确性和完整性。

2. 成本加成定价分析：

- 计算每个蔬菜品类的成本，成本可以包括采购成本、运输成本和其他相关成本。
- 根据成本加成定价方法，确定各蔬菜品类的售价。通常，售价可以设置为成本的加成百分比。

3. 建立优化模型：

目标函数：

- 定义一个目标函数，以最大化商超的总收益为目标。总收益可以通过销售收入减去总成本来计算。

决策变量：

- 对于每个蔬菜品类：
  - 定价决策变量：售价（Price）
  - 补货决策变量：每天的补货数量（Replenishment）

约束条件：

- 约束条件包括：
  - 需求约束：每天的销售数量不得超过补货数量，以避免库存积压或售罄。
  - 补货总量约束：每天的补货总量不能超过一定的限制，以避免过度采购。
  - 售价范围约束：售价不能低于最低限价，也不能高于最高限价。
  - 损耗约束：考虑蔬菜的损耗率，以避免过多的损失。

4. 模型求解：

- 使用线性规划求解器（如 PuLP、Gurobi、CPLEX 等）来求解建立的线性规划模型。这些求解器可以帮助找到最优的售价和补货决策，以最大化总收益。

5. 模型输出与决策：

- 求解器输出最优的售价和补货决策，以及相应的总收益。
- 基于模型的结果，制定未来一周（2023 年 7 月 1-7 日）的日补货总量和定价策略，以最大化商超的收益。

6. 模型评估与调整：

- 对模型的输出进行评估，验证模型的可行性和有效性。
- 如果需要，可以根据实际情况调整模型的参数或约束条件，以优化决策策略。

```
import pandas as pd
import pulp

# 读取销售数据
data = pd.read_csv('sales_data.csv')

# 假设您已经有了每个蔬菜品类的成本数据和损耗率数据
# 请确保数据的格式与您的实际数据一致

# 创建线性规划问题
model = pulp.LpProblem("Maximize_Revenue", pulp.LpMaximize)

# 创建决策变量：售价和补货数量
price = pulp.LpVariable.dicts("Price", data['Category'], lowBound=0, upBound=None, cat='Continuous')
replenishment = pulp.LpVariable.dicts("Replenishment", data['Category'], lowBound=0, upBound=None, cat='Continuous')

# 定义目标函数：最大化总收益
model += pulp.lpSum((price[category] * replenishment[category] * (1 - data['LossRate'][i]) - data['Cost'][i]) * data['Sales'][i]
                    for i, category in enumerate(data['Category']))

# 添加约束条件：需求约束、补货总量约束、售价范围约束
for category, df_category in data.groupby('Category'):
    model += pulp.lpSum(replenishment[category]) >= pulp.lpSum(df_category['Sales']), f"Demand_Constraint_{category}"
    model += pulp.lpSum(replenishment[category]) <= 7 * pulp.lpSum(df_category['Sales']), f"Replenishment_Constraint_{category}"
    model += price[category] >= df_category['Cost'].min(), f"Min_Price_Constraint_{category}"
    model += price[category] <= df_category['Cost'].max(), f"Max_Price_Constraint_{category}"

# 求解线性规划问题
model.solve()
```

```
# 打印结果
print(f"Status: {pulp.LpStatus[model.status]}")

for category in data['Category']:
    print(f"Category: {category}")
    print(f"Optimal Price: ${price[category].varValue:.2f}")
    print(f"Optimal Replenishment: {replenishment[category].varValue:.2f} units")
    print()

# 计算总收益
total_revenue = pulp.value(model.objective)
print(f"Total Revenue: ${total_revenue:.2f}")
```

题目三

问题 3 要求制定单品的补货计划，以控制可售单品总数在 27-33 个之间，同时满足最小陈列量的要求，最大化商超的收益。为了解决这个问题，可以使用混合整数线性规划（Mixed Integer Linear Programming，MILP）来建立优化模型。

1. 数据准备：

- 使用附件 3 中的数据，提取可售单品的信息，包括单品的销售数据、成本、损耗率等。
- 对数据进行清洗和处理，确保数据的准确性和完整性。

2. 建立优化模型：

目标函数：

- 定义一个目标函数，以最大化商超的总收益为目标。总收益可以通过销售收入减去总成本来计算。

决策变量：

- 对于每个可售单品：
    - 订购数量决策变量：订单量（OrderQuantity），是一个整数变量，表示要订购的单品数量。

- 定价决策变量：售价（Price），是一个连续变量，表示单品的售价。

约束条件：

- 约束条件包括：
  - 单品数量约束：总共订购的单品数量必须在 27-33 个之间。
  - 最小陈列量约束：每个单品的订购数量必须满足最小陈列量（2.5 千克）的要求。
  - 售价范围约束：售价不能低于最低限价，也不能高于最高限价。
  - 损耗约束：考虑单品的损耗率，避免过多的损失。

3. 求解优化模型：

- 使用混合整数线性规划求解器（如 PuLP、Gurobi、CPLEX 等）来求解建立的 MILP 模型。
- 在给定的可售单品中，找到最佳的订购数量和定价策略，以最大化总收益。

4. 模型输出与决策：

- 求解器输出最佳的订购数量和定价策略，以及相应的总收益。
- 根据模型的结果，制定 7 月 1 日的单品补货量和定价策略。

5. 模型评估与调整：

- 对模型的输出进行评估，验证模型的可行性和有效性。
- 如果需要，可以根据实际情况调整模型的参数或约束条件，以优化决策策略。

目标函数： 最大化商超的总收益。总收益可以通过销售收入减去总成本来计算。

$$\text{Maximize } \sum_{i} (P_i \cdot Q_i \cdot (1 - \text{损耗率}_i) - \text{成本}_i \cdot Q_i)$$

决策变量： 对于每个可售单品，我们有两个决策变量：

- 订购数量决策变量（整数）：Q_i，表示要订购的单品数量。
- 定价决策变量（连续）：P_i，表示单品的售价。

约束条件：

1. 单品数量约束：总共订购的单品数量必须在 27-33 个之间。 $27 \leq \sum_i Q_i \leq 33$

$$27 \leq \sum_{i} Q_i \leq 33$$

2. 最小陈列量约束：每个单品的订购数量必须满足最小陈列量（2.5 千克）的要求。 $Q_i \geq 2.5, \text{ for all } i$

$$Q_i \geq 2.5, \text{ for all } i$$

3. 售价范围约束：售价不能低于最低限价，也不能高于最高限价。 $P_i \geq \text{最低限价}, \text{ for all } i$

$$P_i \geq \text{最低限价}, \text{ for all } i$$

$$P_i \leq \text{最高限价}, \text{ for all } i$$

$$P_i \leq \text{最高限价}, \text{ for all } i$$

4. 损耗约束：考虑单品的损耗率，避免过多的损失。 $Q_i \leq (1 - \text{损耗率}_i) \times \text{可售库存}_i, \text{ for all } i$

$$Q_i \leq (1 - \text{损耗率}_i) \cdot \text{可售库存}_i, \text{ for all } i$$

目标函数的定义： 最大化总收益，其中总收益由每个单品的销售收入减去成本组成。Maximize $\sum_i (P_i \cdot Q_i \cdot (1 - \text{损耗率}_i) - \text{成本}_i \cdot Q_i)$

```
import pulp

# 数据准备，假设 data 是包含单品信息的 DataFrame
data = pd.read_csv('sales_data.csv')

# 创建线性规划问题
model = pulp.LpProblem("Maximize_Revenue", pulp.LpMaximize)

# 创建决策变量
order_quantities = pulp.LpVariable.dicts("OrderQuantity", data.index,
lowBound=0, cat='Integer')
prices = pulp.LpVariable.dicts("Price", data.index, lowBound=0,
cat='Continuous')

# 目标函数：最大化总收益
model += pulp.lpSum((prices[i] * order_quantities[i] * (1 -
data['LossRate'][i]) - data['Cost'][i] * order_quantities[i])
                    for i in data.index)

# 约束条件
# 1. 单品数量约束
model += 27 <= pulp.lpSum(order_quantities[i] for i in data.index) <=
33

# 2. 最小陈列量约束
for i in data.index:
    model += order_quantities[i] >= 2.5

# 3. 售价范围约束
for i in data.index:
    model += prices[i] >= data['MinPrice'][i]
    model += prices[i] <= data['MaxPrice'][i]

# 4. 损耗约束
for i in data.index:
    model += order_quantities[i] <= (1 - data['LossRate'][i]) *
data['AvailableInventory'][i]

# 求解线性规划问题
```

```
model.solve()

# 打印结果
print(f"Status: {pulp.LpStatus[model.status]}")

for i in data.index:
    print(f"单品编号: {i}")
    print(f"Optimal 订购数量: {order_quantities[i].varValue}")
    print(f"Optimal 售价: {prices[i].varValue}")
    print()

# 计算总收益
total_revenue = pulp.value(model.objective)
print(f"Total Revenue: {total_revenue:.2f}")
```

题目四

1. 实时销售数据：

- 数据类型：每天不同蔬菜品类和单品的销售数量、销售额、销售渠道等数据。
- 帮助：实时销售数据可以提供关于当前需求的信息，帮助商超做出更准确的补货决策和定价策略。它还可以帮助商超跟踪销售趋势和季节性变化。

2. 供应链数据：

- 数据类型：不同蔬菜品类和单品的供应链信息，包括供应商、供应时间、采购价格等。
- 帮助：供应链数据可以帮助商超了解供应商的性能、稳定性和交货准时性，有助于做出更好的供应商选择和采购决策。

3. 成本数据：

- 数据类型：不同蔬菜品类和单品的采购成本、运输成本、仓储成本等。

- 帮助：成本数据对于制定成本加成定价策略非常重要。它们可以帮助商超确保售价足够高，以覆盖所有成本并获得合理的利润。

4. 损耗率数据：

- 数据类型：不同蔬菜品类和单品的损耗率数据。
- 帮助：损耗率数据可以帮助商超优化库存管理和补货决策。它们还有助于减少损失并提高利润率。

5. 市场趋势数据：

- 数据类型：市场趋势、竞争对手价格、消费者偏好等数据。
- 帮助：市场趋势数据可以帮助商超调整定价策略，以保持竞争力。了解竞争对手的价格和消费者偏好有助于制定更具吸引力的定价策略。

6. 季节性数据：

- 数据类型：不同季节蔬菜销售数据，季节性价格变动数据等。
- 帮助：季节性数据有助于商超在不同季节制定不同的补货和定价策略，以满足季节性需求和价格波动。

7. 库存数据：

- 数据类型：库存量、库存周转率等数据。
- 帮助：库存数据可以帮助商超优化库存管理，减少库存积压或售罄的情况，提高资金利用率。

8. 顾客反馈数据：

- 数据类型：顾客反馈、投诉和建议等数据。
- 帮助：顾客反馈数据可以帮助商超了解顾客满意度和需求变化，有助于改进产品选择和服务质量。

```
#1. 数据采集：

#首先，您需要确定数据的来源，可能是数据库、API、网页抓取等。然后，使用适当的 Python 库来获取数据。
import requests
import pandas as pd
```

```
# 从 API 获取数据的示例
url = 'https://api.example.com/data'
response = requests.get(url)
data = response.json()  # 将响应数据解析为 JSON 格式

# 将数据存储为 DataFrame

df = pd.DataFrame(data)

#2. 数据清洗和预处理：

#对数据进行清洗和预处理，包括处理缺失值、重复值、异常值等。
# 处理缺失值
df.dropna(inplace=True)

# 处理重复值
df.drop_duplicates(inplace=True)

# 处理异常值
# 例如，移除销售数量小于 0 的记录
df = df[df['Sales'] >= 0]



#3. 数据分析：

#使用 pandas 和其他数据分析库来执行数据分析任务，如统计、聚合、计算指标等。
# 统计数据摘要
summary_stats = df.describe()

# 计算总销售额
total_sales = df['Sales'].sum()

# 分组聚合操作
category_sales = df.groupby('Category')['Sales'].sum()

# 数据可视化
import matplotlib.pyplot as plt

# 绘制销售量的直方图
plt.hist(df['Sales'], bins=20, color='blue', edgecolor='black')
plt.xlabel('Sales')
plt.ylabel('Frequency')
```

```
plt.title('Sales Distribution')
plt.show()
#4. 数据可视化:

#使用数据可视化库（如 matplotlib、seaborn）来创建图表和可视化工具，以更好地理解数据。
# 绘制销售量的折线图
plt.plot(df['Date'], df['Sales'], marker='o', linestyle='-', color='green')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Sales Over Time')
plt.xticks(rotation=45)
plt.show()

```