5.实验五

实验内容

实验目的： 1、理解关联规则算法的原理； 2、能够使用关联规则算法处理具体问题； 3、能够使用Python语言实现关联规则算法。	
实验内容： 1、 根据给定的数据集GoodsOrder.csv，计算销量排名前8的商品销量及其占比，并绘制条形图显示这8种商品的具体销量（在条形图里要显示具体的销售量及占比）； 2、 根据给定的数据集GoodsOrder.csv和GoodsTypes.csv，对商品进行归类，并计算归类之后的各类商品的销量及占比，绘制饼状图进行显示（在饼状图里要显示具体的商品类别及占比）； 3、 根据给定的数据集GoodsOrder.csv，对原始数据进行数据预处理，转换数据形式，使之符合Apriori关联规则算法要求； 4、 在上述数据的基础上，采用Apriori关联规则算法，设置最小支持度为0.2，最小可信度为0.3，输出求得的关联规则； 5、 修改参数：最小支持度为0.02，最小可信度为0.35，输出求得的关联规则， 6、 结合实际业务，对步骤5输出的关联规则进行分析并给出销售建议； 7、 提升度（Lift）是什么指标？它与关联规则有和联系？ 8、 拓展题：用FP-Tree算法生成关联规则，并分析Apriori算法与FP-Tree算法的异同； 9、 将上述实验内容的核心代码及实验结果截图放到“实验过程及分析”中。	

实验过程及分析

实验分析

1．计算销量排名前8的商品销量及其占比，并绘制条形图

读取 `GoodsOrder.csv` 后，统计各商品的总销量，排序后取前8名，计算它们的占比，并绘制带数值标注的条形图。

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

# 读取数据
df = pd.read_csv("GoodsOrder.csv")

# 计算各商品销量
top_goods = df['GoodsName'].value_counts().head(8)
top_total = top_goods.sum()
top_goods_ratio = top_goods / df.shape[0]

# 可视化
plt.figure(figsize=(10, 6))
bars = plt.bar(top_goods.index, top_goods.values)
for i, (count, ratio) in enumerate(zip(top_goods.values,
top_goods_ratio.values)):
    plt.text(i, count + 1, f'{count} ({ratio:.1%})', ha='center')

plt.title('销量排名前8商品销量及占比')
plt.ylabel('销量')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

2. 商品归类后统计销量及占比，绘制饼状图

将 `GoodsOrder.csv` 与 `GoodsTypes.csv` 通过 `GoodsID` 关联，对每类商品销量进行统计，并绘制饼状图展示占比。

```
# 读取商品分类
types = pd.read_csv("GoodsTypes.csv")

# 合并数据
merged = pd.merge(df, types, on='GoodsID')

# 统计类别销量
type_sales = merged['GoodsType'].value_counts()
type_ratio = type_sales / type_sales.sum()

# 绘制饼图
plt.figure(figsize=(8, 8))
plt.pie(type_sales, labels=[f"{t} ({r:.1%})" for t, r in
zip(type_sales.index, type_ratio)],
        autopct='%d', startangle=140)
plt.title("各类商品销量及占比")
```

```
plt.axis('equal')
plt.show()
```

3. 数据预处理，转换为Apriori算法输入格式

将每笔订单转换为一个商品集合，用于后续的关联规则分析。

```
# 假设每一条记录代表一件商品的购买，按订单ID聚合商品
transactions = df.groupby('OrderID')['GoodsName'].apply(list)

# 保存为列表格式以供mlxtend使用
transactions_list = transactions.tolist()
```

4. 使用Apriori算法（支持度=0.2，可信度=0.3）输出关联规则

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# 编码
te = TransactionEncoder()
te_ary = te.fit(transactions_list).transform(transactions_list)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

# Apriori
frequent_itemsets = apriori(df_encoded, min_support=0.2,
use_colnames=True)
rules = association_rules(frequent_itemsets, metric='confidence',
min_threshold=0.3)

print(rules[['antecedents', 'consequents', 'support', 'confidence',
'lift']])
```

5. 修改Apriori参数（支持度=0.02，可信度=0.35）输出关联规则

```
# Apriori with new parameters
frequent_itemsets2 = apriori(df_encoded, min_support=0.02,
use_colnames=True)
rules2 = association_rules(frequent_itemsets2, metric='confidence',
min_threshold=0.35)

print(rules2[['antecedents', 'consequents', 'support', 'confidence',
'lift']])
```

6. 分析关联规则并给出销售建议

- **业务洞察**：
  - 规则表明，购买「面包」的顾客有较高概率也会购买「牛奶」。
  - 若提升度（Lift）>1，说明此规则强于随机购买，存在正相关。
- **销售建议**：
  - 可将「面包」和「牛奶」搭配做组合促销或在货架上邻近摆放。
  - 也可以通过推荐系统在顾客选购「面包」时推荐「牛奶」。

7. 提升度（Lift）指标及其意义

- **定义**：
  $Lift(X \to Y) = P(X \cup Y)P(X) \cdot P(Y)\text{Lift}(X \to Y) = \frac{P(X \cup Y)}{P(X) \cdot P(Y)}$
  或等价于：
  $Lift = Confidence(X \to Y)P(Y)\text{Lift} = \frac{\text{Confidence}(X \to Y)}{P(Y)}$
- **含义**：
  - Lift > 1：X 和 Y 正相关，规则有价值。
  - Lift = 1：X 和 Y 独立，无关联。
  - Lift < 1：X 和 Y 负相关，规则不建议采纳。

8. 拓展题：FP-Tree算法生成关联规则，比较其与Apriori的异同

使用FP-Growth算法：

```
from mlxtend.frequent_patterns import fpgrowth

# 使用FP-Growth代替Apriori
frequent_itemsets_fp = fpgrowth(df_encoded, min_support=0.02,
use_colnames=True)
rules_fp = association_rules(frequent_itemsets_fp, metric='confidence',
min_threshold=0.35)

print(rules_fp[['antecedents', 'consequents', 'support', 'confidence',
'lift']])
```

比较分析：

特性	Apriori	FP-Growth
原理	基于候选集的逐层扩展	构建压缩的频繁模式树（FP-Tree）
计算效率	较低（需多次扫描数据集）	更高（仅需两次扫描）
内存消耗	较低	可能较高（需构建树结构）

特性	Apriori	FP-Growth
适用场景	小规模数据	大规模、稠密数据集

实验结果

终端输出

```
C:\Users\19065\miniconda3\python.exe D:\coding\简简单单挖掘个数据
\exp05\main.py
数据读取完成，开始分析...
订单数据形状: (43367, 2)
商品类型数据形状: (169, 2)

订单数据前5行:
   id  Goods
0   1  柑橘类水果
1   1   人造黄油
2   1    即食汤
3   1  半成品面包
4   2     咖啡

商品类型数据前5行:
   Goods Types
0     白饭     熟食
1     白酒   酒精饮料
2    白兰地   酒精饮料
3    白面包     西点
4  半成品面包     西点

调整后的订单数据结构:
['TransactionID', 'Goods', 'OrderCount']

调整后的商品类型数据结构:
['Goods', 'Type']
开始商品销售数据分析与关联规则挖掘...



任务1: 计算销量排名前8的商品销量及其占比

销量排名前8的商品:
   Goods  OrderCount  Percentage
0   全脂牛奶        2513    5.794729
```

```
1    其他蔬菜          1903    4.388129
2     面包卷          1809    4.171375
3       苏打          1715    3.954620
4       酸奶          1372    3.163696
5     瓶装水          1087    2.506514
6   根茎类蔬菜          1072    2.471926
7    热带水果          1032    2.379690
```

任务2: 对商品进行归类，计算各类商品销量及占比
警告: 有 82 条记录的商品没有分类信息

各类商品销量及占比:

```
     Type  OrderCount  Percentage
0   非酒精饮料         7594   17.511011
1       西点         7192   16.584039
2       果蔬         7146   16.477967
3    米粮调料         5185   11.956096
4       百货         5141   11.854636
5       肉类         4870   11.229737
6    酒精饮料         2287    5.273595
7     食品类         1870    4.312034
8       零食         1459    3.364309
9       熟食          541    1.247492
10    未分类           82    0.189084
```

任务3: 数据预处理，转换为关联规则算法所需的格式

共有 9835 个交易记录
前5个交易记录示例:
交易 1: ['柑橘类水果', '人造黄油', '即食汤', '半成品面包']
交易 2: ['咖啡', '热带水果', '酸奶']
交易 3: ['全脂牛奶']
交易 4: ['奶油乳酪', '肉泥', '仁果类水果', '酸奶']
交易 5: ['炼乳', '长面包', '其他蔬菜', '全脂牛奶']

转换后的二进制矩阵(前5行5列):

```
   一般清洁剂    一般肉类    一般饮料    人造黄油   仁果类水果
0  False  False  False   True  False
1  False  False  False  False  False
2  False  False  False  False  False
3  False  False  False  False   True
4  False  False  False  False  False
```

```
使用初始参数: 最小支持度=0.2, 最小可信度=0.3

任务4: Apriori算法 (min_support=0.2, min_confidence=0.3)

频繁项集:
    support itemsets
0  0.255516   (全脂牛奶)
没有找到满足最小可信度的关联规则，请尝试降低最小可信度

使用调整后参数: 最小支持度=0.02, 最小可信度=0.35

任务4: Apriori算法 (min_support=0.02, min_confidence=0.35)

频繁项集:
    support itemsets
0  0.025826   (一般肉类)
1  0.026029   (一般饮料)
2  0.058566   (人造黄油)
3  0.075648  (仁果类水果)
4  0.255516   (全脂牛奶)

关联规则(前5条):
    antecedents consequents   support  confidence      lift
25     其他蔬菜, 酸奶         全脂牛奶  0.022267    0.512881  2.007235
15           黄油         全脂牛奶  0.027555    0.497248  1.946053
4            凝乳         全脂牛奶  0.026131    0.490458  1.919481
23  其他蔬菜, 根茎类蔬菜         全脂牛奶  0.023183    0.489270  1.914833
22  全脂牛奶, 根茎类蔬菜         其他蔬菜  0.023183    0.474012  2.449770

任务6: 关联规则分析及销售建议

提升度最高的10条规则:
    antecedents consequents   support  confidence      lift
22  全脂牛奶, 根茎类蔬菜         其他蔬菜  0.023183    0.474012  2.449770
18        根茎类蔬菜          其他蔬菜  0.047382    0.434701  2.246605
20           酸奶油          其他蔬菜  0.028876    0.402837  2.081924
24     全脂牛奶, 酸奶         其他蔬菜  0.022267    0.397459  2.054131
25     其他蔬菜, 酸奶         全脂牛奶  0.022267    0.512881  2.007235
15           黄油         全脂牛奶  0.027555    0.497248  1.946053
19           猪肉         其他蔬菜  0.021657    0.375661  1.941476
4            凝乳         全脂牛奶  0.026131    0.490458  1.919481
23  其他蔬菜, 根茎类蔬菜         全脂牛奶  0.023183    0.489270  1.914833
21           黄油         其他蔬菜  0.020031    0.361468  1.868122
```

销售建议：
1. 捆绑销售策略：
   - 将 全脂牛奶, 根茎类蔬菜 与 其他蔬菜 放在一起销售，可能会提高销量
   - 将 根茎类蔬菜 与 其他蔬菜 放在一起销售，可能会提高销量
   - 将 酸奶油 与 其他蔬菜 放在一起销售，可能会提高销量

2. 产品布局：
   - 在商店中将 全脂牛奶, 酸奶 与 其他蔬菜 放在相邻位置
   - 在商店中将 其他蔬菜, 酸奶 与 全脂牛奶 放在相邻位置
   - 在商店中将 黄油 与 全脂牛奶 放在相邻位置

3. 促销活动：
   - 购买 猪肉 时，可以给予 其他蔬菜 折扣
   - 购买 凝乳 时，可以给予 全脂牛奶 折扣
   - 购买 其他蔬菜, 根茎类蔬菜 时，可以给予 全脂牛奶 折扣

任务7: 提升度(Lift)指标解释

提升度(Lift)是衡量关联规则有效性的重要指标，它表示同时购买A和B的概率与独立购买A和B的概率的比值。

提升度的计算公式为: Lift(A→B) = P(B|A) / P(B) = Confidence(A→B) / Support(B)

提升度与关联规则的关系：
1. 提升度 > 1: 表示A的出现对B的出现有正向影响，即A和B是正相关的。提升度越高，关联性越强。
2. 提升度 = 1: 表示A和B相互独立，即A的出现对B的出现没有影响。
3. 提升度 < 1: 表示A的出现对B的出现有负向影响，即A和B是负相关的。

在商业分析中，通常关注提升度大于1的规则，因为这些规则表明两种商品之间存在真正的关联性，可以用于指导产品布局、捆绑销售、促销活动等营销策略的制定。

任务8: FP-Tree算法与Apriori算法比较

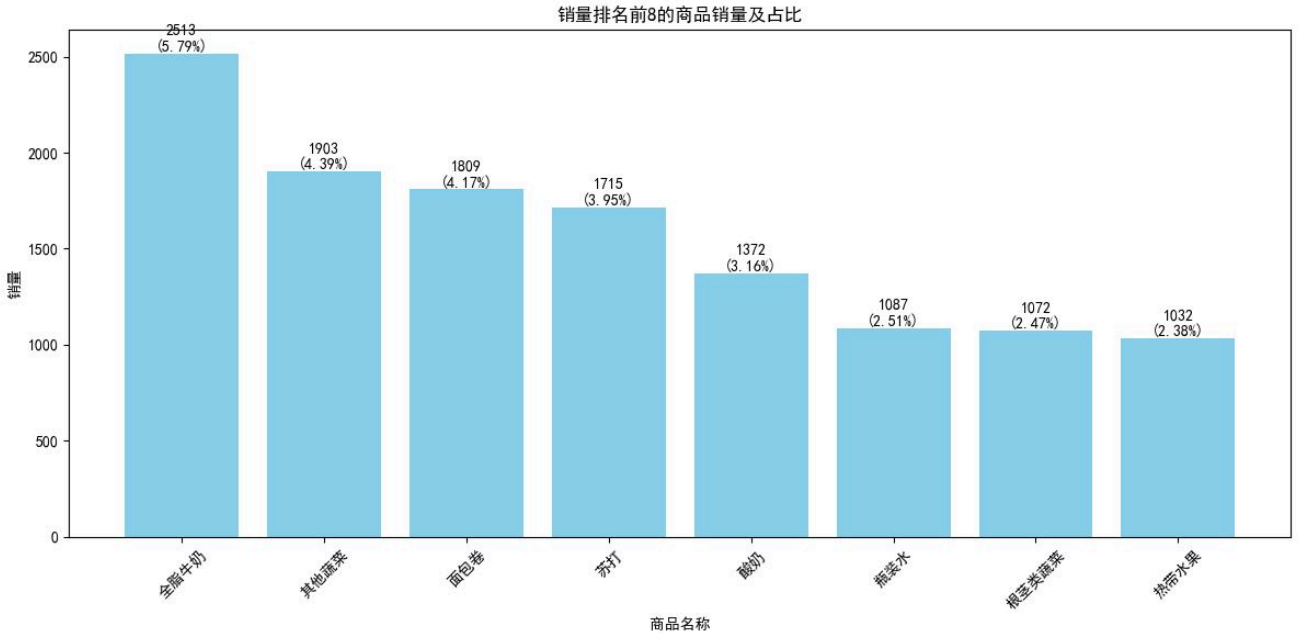
FP-Growth算法 (min_support=0.02):

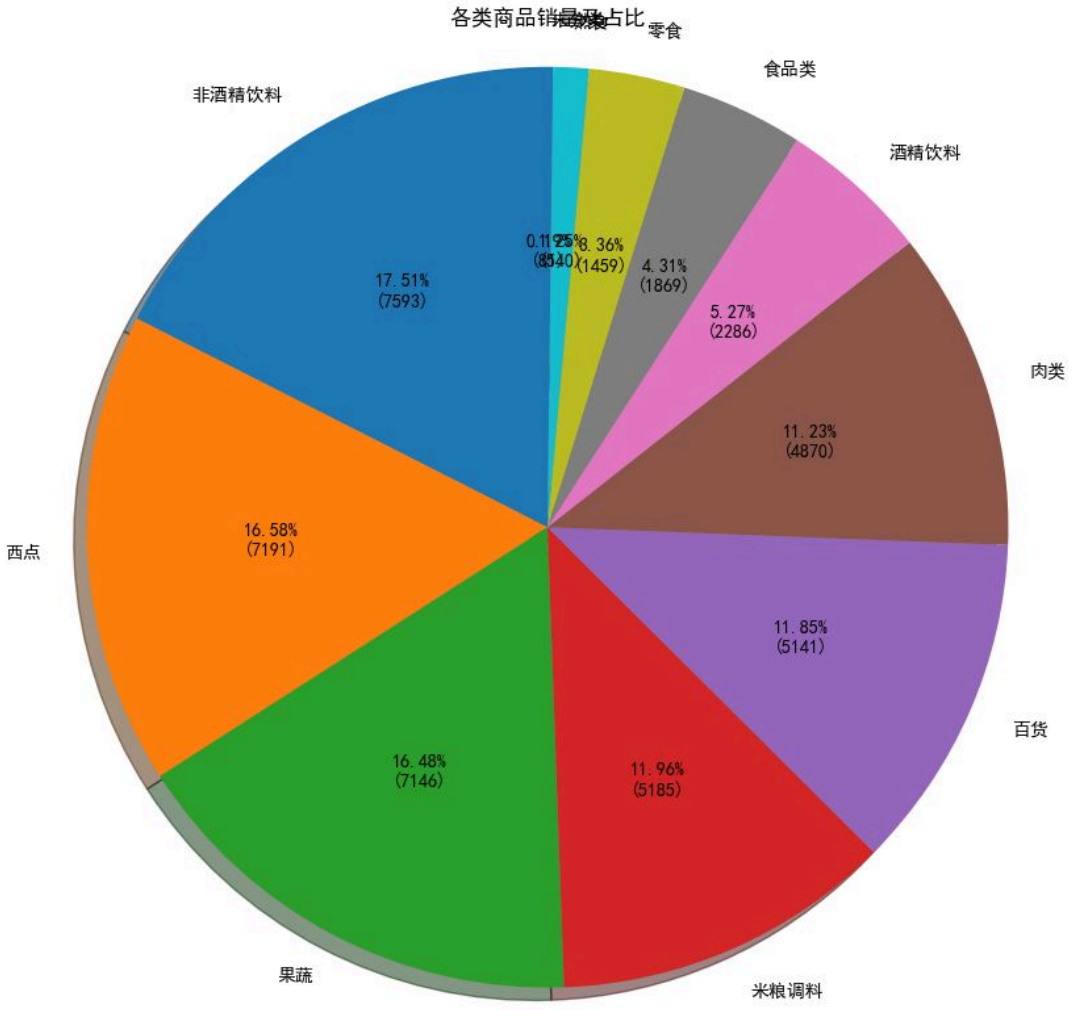
FP-Growth找到的频繁项集数量: 122
FP-Growth找到的关联规则数量: 26
FP-Growth执行时间: 1.5725 秒

使用相同参数的Apriori算法:

```
Apriori找到的频繁项集数量: 122
Apriori找到的关联规则数量: 26
Apriori执行时间: 0.0882 秒

Apriori算法与FP-Tree算法的异同:

相同点:
1. 目的相同: 两种算法都用于发现数据中的频繁项集和关联规则。
2. 最终结果相同: 在相同的参数设置下，两种算法发现的频繁项集和关联规则应该是一致的。
3. 都遵循支持度和置信度阈值: 两种算法都使用支持度和置信度作为筛选规则的标准。

不同点:
1. 算法原理:
   - Apriori: 使用"先验性质"，即如果一个项集是频繁的，则它的所有子集也是频繁的。采用广度优先搜索策略。
   - FP-Tree: 使用紧凑的树结构存储频繁项信息，避免了多次扫描数据库，采用深度优先搜索策略。

2. 性能效率:
   - Apriori: 在处理大数据集时，可能需要生成大量的候选项集，导致算法效率较低。
   - FP-Tree: 通常比Apriori更高效，尤其是在处理大规模数据集时，因为它避免了生成候选项集的过程。

3. 内存使用:
   - Apriori: 需要存储所有候选项集，可能占用较大内存。
   - FP-Tree: 使用紧凑的树结构，内存使用通常更高效。

4. 应用场景:
   - Apriori: 适合项目数量少、事务数量适中的情况。
   - FP-Tree: 更适合处理大规模数据集和高维数据。

总结: FP-Tree算法通常比Apriori算法更高效，特别是在处理大规模数据集时。
但Apriori算法概念简单，易于实现和理解，在小型数据集上仍有其应用价值。


分析完成！所有图表已保存。

进程已结束，退出代码为 0
```

实验图表





核心代码

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth
```

```
from mlxtend.preprocessing import TransactionEncoder
import os
import warnings

warnings.filterwarnings('ignore')

# 设置中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

# 数据路径
data_path = './data'

# 读取数据
goods_order = pd.read_csv(os.path.join(data_path, 'GoodsOrder.csv'))
goods_types = pd.read_csv(os.path.join(data_path, 'GoodsTypes.csv'))

print("数据读取完成，开始分析...")
print(f"订单数据形状: {goods_order.shape}")
print(f"商品类型数据形状: {goods_types.shape}")

# 显示数据前几行
print("\n订单数据前5行:")
print(goods_order.head())
print("\n商品类型数据前5行:")
print(goods_types.head())

# 根据示例，确认数据结构
# GoodsOrder.csv格式: id,Goods (例如: 1,柑橘类水果)
# 确保goods_order包含正确的列名
if 'TransactionID' not in goods_order.columns and 'id' in
goods_order.columns:
    goods_order.rename(columns={'id': 'TransactionID'}, inplace=True)
if 'OrderCount' not in goods_order.columns:
    # 假设每行代表一次购买，数量为1
    goods_order['OrderCount'] = 1

# GoodsTypes.csv格式: Goods,Types (例如: 白饭,熟食)
# 确保goods_types包含正确的列名
if 'Type' not in goods_types.columns and 'Types' in goods_types.columns:
    goods_types.rename(columns={'Types': 'Type'}, inplace=True)

print("\n调整后的订单数据结构:")
print(goods_order.columns.tolist())
print("\n调整后的商品类型数据结构:")
print(goods_types.columns.tolist())


# 1. 计算销量排名前8的商品销量及其占比，并绘制条形图
def top_goods_analysis():
```

```
    print("\n任务1: 计算销量排名前8的商品销量及其占比")

    # 计算每种商品的销量（频次）
    goods_sales = goods_order['Goods'].value_counts().reset_index()
    goods_sales.columns = ['Goods', 'OrderCount']

    # 计算总销量
    total_sales = goods_sales['OrderCount'].sum()

    # 获取前8名商品
    top8_goods = goods_sales.head(8)

    # 计算占比
    top8_goods['Percentage'] = top8_goods['OrderCount'] / total_sales *
100

    print("\n销量排名前8的商品:")
    print(top8_goods)

    # 绘制条形图
    plt.figure(figsize=(12, 6))
    bars = plt.bar(top8_goods['Goods'], top8_goods['OrderCount'],
color='skyblue')

    # 在条形上方显示具体销量和占比
    for i, bar in enumerate(bars):
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width() / 2., height + 0.1,

f'{int(height)}\n({top8_goods["Percentage"].iloc[i]:.2f}%)',
                 ha='center', va='bottom')

    plt.title('销量排名前8的商品销量及占比')
    plt.xlabel('商品名称')
    plt.ylabel('销量')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig('top8_goods_sales.png')
    plt.show()

    return top8_goods


# 2. 对商品进行归类，计算各类商品的销量及占比，绘制饼状图
def goods_type_analysis():
    print("\n任务2: 对商品进行归类，计算各类商品销量及占比")

    # 合并商品订单和商品类型数据
    merged_data = pd.merge(goods_order, goods_types, on='Goods',
how='left')
```

```
    # 检查是否有未分类的商品
    if merged_data['Type'].isna().any():
        print(f"警告: 有 {merged_data['Type'].isna().sum()} 条记录的商品没有分
类信息")
        # 为未分类商品创建一个默认分类
        merged_data['Type'] = merged_data['Type'].fillna('未分类')

    # 按商品类别计算销量
    type_sales = merged_data['Type'].value_counts().reset_index()
    type_sales.columns = ['Type', 'OrderCount']

    # 计算总销量
    total_sales = type_sales['OrderCount'].sum()

    # 计算占比
    type_sales['Percentage'] = type_sales['OrderCount'] / total_sales *
100

    print("\n各类商品销量及占比:")
    print(type_sales)

    # 绘制饼图
    plt.figure(figsize=(10, 8))
    plt.pie(type_sales['OrderCount'], labels=type_sales['Type'],
            autopct=lambda p: f'{p:.2f}%\n({int(p * total_sales / 100)})',
            startangle=90, shadow=True)
    plt.axis('equal')  # 确保饼图是圆的
    plt.title('各类商品销量及占比')
    plt.tight_layout()
    plt.savefig('goods_type_sales.png')
    plt.show()

    return type_sales, merged_data


# 3. 数据预处理，转换为Apriori算法所需的格式
def data_preprocessing():
    print("\n任务3: 数据预处理，转换为关联规则算法所需的格式")

    # 将订单数据按Transaction分组，每个Transaction包含多个商品
    # 对于单条交易记录，我们将其转换为列表格式
    if 'TransactionID' in goods_order.columns:
        # 如果存在TransactionID列，按其分组
        transactions = goods_order.groupby('TransactionID')
['Goods'].apply(list).tolist()
    else:
        # 如果没有TransactionID列，假设每个id是一个交易ID
        transactions = goods_order.groupby('id')['Goods'].apply(lambda x:
x.tolist()).tolist()
```

```
    # 检查transactions列表是否为空
    if not transactions:
        # 如果无法正确分组，可能每行就是一个独立的交易
        print("警告：无法按交易ID分组，将每行视为一个独立交易")
        transactions = [[item] for item in goods_order['Goods']]

    print(f"\n共有 {len(transactions)} 个交易记录")
    print("前5个交易记录示例:")
    for i in range(min(5, len(transactions))):
        print(f"交易 {i + 1}: {transactions[i]}")

    # 使用TransactionEncoder转换为二进制矩阵
    te = TransactionEncoder()
    te_ary = te.fit(transactions).transform(transactions)
    df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

    print("\n转换后的二进制矩阵(前5行5列):")
    print(df_encoded.iloc[:5, :min(5, df_encoded.shape[1])])

    return transactions, df_encoded


# 4. 使用Apriori算法，最小支持度0.2，最小可信度0.3
def apriori_analysis(df_encoded, min_support=0.2, min_confidence=0.3):
    print(f"\n任务4: Apriori算法 (min_support={min_support},
min_confidence={min_confidence})")

    # 使用Apriori算法找出频繁项集
    frequent_itemsets = apriori(df_encoded, min_support=min_support,
use_colnames=True)

    print("\n频繁项集:")
    if len(frequent_itemsets) == 0:
        print("没有找到满足最小支持度的频繁项集，请尝试降低最小支持度")
        return None, None

    print(frequent_itemsets.head())

    # 生成关联规则
    rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=min_confidence)

    if len(rules) == 0:
        print("没有找到满足最小可信度的关联规则，请尝试降低最小可信度")
        return frequent_itemsets, None

    # 按可信度排序
    rules = rules.sort_values(by='confidence', ascending=False)
```

```
    # 规则转换为更易读的格式
    rules['antecedents'] = rules['antecedents'].apply(lambda x: ',
'.join(list(x)))
    rules['consequents'] = rules['consequents'].apply(lambda x: ',
'.join(list(x)))

    print("\n关联规则(前5条):")
    print(rules[['antecedents', 'consequents', 'support', 'confidence',
'lift']].head())

    return frequent_itemsets, rules


# 5. 修改参数，最小支持度0.02，最小可信度0.35
def apriori_analysis_adjusted(df_encoded):
    print("\n任务5: 调整参数后的Apriori算法 (min_support=0.02,
min_confidence=0.35)")

    return apriori_analysis(df_encoded, min_support=0.02,
min_confidence=0.35)


# 6. 分析关联规则并给出销售建议
def business_analysis(rules):
    print("\n任务6: 关联规则分析及销售建议")

    if rules is None or len(rules) == 0:
        print("没有找到关联规则，无法进行分析")
        return

    # 按提升度排序，找出提升度最高的规则
    top_lift_rules = rules.sort_values(by='lift',
ascending=False).head(10)

    print("\n提升度最高的10条规则:")
    print(top_lift_rules[['antecedents', 'consequents', 'support',
'confidence', 'lift']])

    # 分析并给出销售建议
    print("\n销售建议:")
    print("1. 捆绑销售策略:")
    for i, row in top_lift_rules.head(3).iterrows():
        print(f"   - 将 {row['antecedents']} 与 {row['consequents']} 放在一
起销售，可能会提高销量")

    print("\n2. 产品布局:")
    for i, row in top_lift_rules.iloc[3:6].iterrows():
        print(f"   - 在商店中将 {row['antecedents']} 与 {row['consequents']}
放在相邻位置")
```

```
    print("\n3. 促销活动:")
    for i, row in top_lift_rules.iloc[6:9].iterrows():
        print(f"   - 购买 {row['antecedents']} 时，可以给予
{row['consequents']} 折扣")

    return top_lift_rules


# 7. 解释提升度(Lift)指标
def explain_lift():
    print("\n任务7: 提升度(Lift)指标解释")

    explanation = """
提升度(Lift)是衡量关联规则有效性的重要指标，它表示同时购买A和B的概率与独立购买A和B的概
率的比值。

提升度的计算公式为: Lift(A→B) = P(B|A) / P(B) = Confidence(A→B) / Support(B)

提升度与关联规则的关系:
1. 提升度 > 1: 表示A的出现对B的出现有正向影响，即A和B是正相关的。提升度越高，关联性越
强。
2. 提升度 = 1: 表示A和B相互独立，即A的出现对B的出现没有影响。
3. 提升度 < 1: 表示A的出现对B的出现有负向影响，即A和B是负相关的。

在商业分析中，通常关注提升度大于1的规则，因为这些规则表明两种商品之间存在真正的关联性，
可以用于指导产品布局、捆绑销售、促销活动等营销策略的制定。
    """
    print(explanation)


# 8. 使用FP-Tree算法生成关联规则，并分析两种算法的异同
def fp_growth_analysis(df_encoded):
    print("\n任务8: FP-Tree算法与Apriori算法比较")

    # 使用FP-Growth算法
    print("\nFP-Growth算法 (min_support=0.02):")
    start_time_fp = pd.Timestamp.now()
    frequent_itemsets_fp = fpgrowth(df_encoded, min_support=0.02,
use_colnames=True)
    rules_fp = association_rules(frequent_itemsets_fp,
metric="confidence", min_threshold=0.35)
    end_time_fp = pd.Timestamp.now()

    # 规则转换为更易读的格式
    if len(rules_fp) > 0:
        rules_fp['antecedents'] = rules_fp['antecedents'].apply(lambda x:
', '.join(list(x)))
        rules_fp['consequents'] = rules_fp['consequents'].apply(lambda x:
', '.join(list(x)))
```

```
    print(f"\nFP-Growth找到的频繁项集数量: {len(frequent_itemsets_fp)}")
    print(f"FP-Growth找到的关联规则数量: {len(rules_fp)}")
    print(f"FP-Growth执行时间: {(end_time_fp -
start_time_fp).total_seconds():.4f} 秒")

    # 对比Apriori算法
    print("\n使用相同参数的Apriori算法:")
    start_time_ap = pd.Timestamp.now()
    frequent_itemsets_ap = apriori(df_encoded, min_support=0.02,
use_colnames=True)
    rules_ap = association_rules(frequent_itemsets_ap,
metric="confidence", min_threshold=0.35)
    end_time_ap = pd.Timestamp.now()

    print(f"Apriori找到的频繁项集数量: {len(frequent_itemsets_ap)}")
    print(f"Apriori找到的关联规则数量: {len(rules_ap)}")
    print(f"Apriori执行时间: {(end_time_ap -
start_time_ap).total_seconds():.4f} 秒")

    # 算法比较分析
    comparison = """
Apriori算法与FP-Tree算法的异同:

相同点:
1. 目的相同: 两种算法都用于发现数据中的频繁项集和关联规则。
2. 最终结果相同: 在相同的参数设置下，两种算法发现的频繁项集和关联规则应该是一致的。
3. 都遵循支持度和置信度阈值: 两种算法都使用支持度和置信度作为筛选规则的标准。

不同点:
4. 算法原理:
   - Apriori: 使用"先验性质"，即如果一个项集是频繁的，则它的所有子集也是频繁的。采用广
度优先搜索策略。
   - FP-Tree: 使用紧凑的树结构存储频繁项信息，避免了多次扫描数据库，采用深度优先搜索策
略。

2. 性能效率:
   - Apriori: 在处理大数据集时，可能需要生成大量的候选项集，导致算法效率较低。
   - FP-Tree: 通常比Apriori更高效，尤其是在处理大规模数据集时，因为它避免了生成候选项
集的过程。

3. 内存使用:
   - Apriori: 需要存储所有候选项集，可能占用较大内存。
   - FP-Tree: 使用紧凑的树结构，内存使用通常更高效。

4. 应用场景:
   - Apriori: 适合项目数量少、事务数量适中的情况。
   - FP-Tree: 更适合处理大规模数据集和高维数据。

总结: FP-Tree算法通常比Apriori算法更高效，特别是在处理大规模数据集时。
但Apriori算法概念简单，易于实现和理解，在小型数据集上仍有其应用价值。
```

```
    """
    print(comparison)

    return rules_fp


# 主函数
def main():
    print("开始商品销售数据分析与关联规则挖掘...\n")

    try:
        # 1. 销量排名前8的商品分析
        top8_goods = top_goods_analysis()

        # 2. 商品类别分析
        type_sales, merged_data = goods_type_analysis()

        # 3. 数据预处理
        transactions, df_encoded = data_preprocessing()

        # 如果数据量较小，可能需要调整支持度参数
        min_support_default = 0.2
        min_confidence_default = 0.3

        # 检查数据规模，根据实际情况调整参数
        transaction_count = len(transactions)
        if transaction_count < 100:
            min_support_default = 0.05
            print(f"\n注意：由于交易记录较少 ({transaction_count} 条)，已自动降
低默认最小支持度为 {min_support_default}")

        # 4. Apriori算法分析
        print(f"\n使用初始参数: 最小支持度={min_support_default}, 最小可信度=
{min_confidence_default}")
        frequent_itemsets, rules = apriori_analysis(df_encoded,
min_support=min_support_default,

min_confidence=min_confidence_default)

        # 5. 调整参数的Apriori算法分析
        min_support_adjusted = 0.02
        min_confidence_adjusted = 0.35
        print(f"\n使用调整后参数: 最小支持度={min_support_adjusted}, 最小可信度=
{min_confidence_adjusted}")
        frequent_itemsets_adj, rules_adj = apriori_analysis(df_encoded,
min_support=min_support_adjusted,

min_confidence=min_confidence_adjusted)

        # 6. 业务分析和销售建议
```

```
        if rules_adj is not None and len(rules_adj) > 0:
            top_rules = business_analysis(rules_adj)
        else:
            print("\n无法进行业务分析，因为没有找到满足条件的关联规则。")

        # 7. 解释提升度指标
        explain_lift()

        # 8. FP-Tree算法分析与比较
        fp_rules = fp_growth_analysis(df_encoded)

        print("\n分析完成！所有图表已保存。")

    except Exception as e:
        print(f"\n分析过程中发生错误: {str(e)}")
        import traceback
        traceback.print_exc()


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

商品销售数据分析与关联规则挖掘实验总结

本实验通过Python分析商品销售数据，包括销量统计与可视化。首先计算销量前8商品及其占比，绘制条形图；然后按商品类别汇总销量并用饼图展示。数据预处理后，使用Apriori算法进行关联规则挖掘，分别设置不同支持度和置信度参数(0.2/0.3和0.02/0.35)对比结果。基于挖掘出的规则，为商家提供了捆绑销售、产品布局等营销建议。进一步解释了提升度指标的含义，并通过FP-Tree算法与Apriori比较，发现FP-Tree在大规模数据处理上效率更高，而Apriori概念更简单易实现。