实验总分：105（除内存分析外都完成）

实验说明：通过 Data 类处理原始数据，得到类成员属性二维数组 data，作为 Apriori 类初始化的条件。

重要方法

通过 Apriori 类的 set_support(self, support, method=1)方法设置支持度，默认方法为剪枝 1，可以选 0，1，2，3，0 为未剪枝的算法，1，2，3 为对应的剪枝算法。

通过 set_relateRules(self, minC, support=support)方法设置置信度。

通过 daxiangxiangji(self, n)方法获得指定的一项集，二项集……n 项集

重要属性

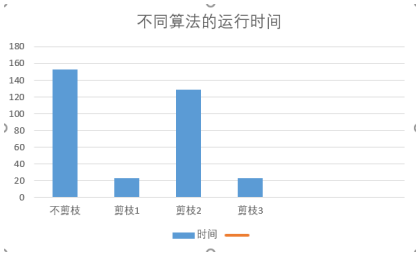
support 支持度

minC = 置信度

ans_C 平凡项集，需要设置支持度后才有

relateRules 关联规则，需要设置置信度后才有

实验分析



在本次实验中最快的为剪枝 1 和剪枝 3，剪枝 2 优势巨大，但是相比 fpgrowth 还是有一点差距，但相对不剪枝还是具有优势。当数据集超过本次实验的九千条数据，剪枝 3 的性能理论上应该由于剪枝 1。

Pyfpgrowth 的调用过程中发现，这个包求出来的频繁项和关联规则都比正确的要少。

实验基本要求（85 分）

1.数据处理（15 分）

通过 Data 类可以直接加载文档，数据处理结果为 Data 类的 data 属性并保存为“data.pk”文件中，商品与数字的对应关系保存在“对应关系.json”文件中。

2.算法实现（50 分）

第三部分，手动实现 dummyApriori 算法-20 分

第四部分，手动实现 advanced Apriori 算法的前两种剪枝策略-10*2=20 分

```
set_support(self, support, method=2)
```

method 方法可以取值为 0，1，2，3

其中 method 对应 dummyApriori，1，2，3 对应三种剪枝策略

调用库 pyfpgrowth 实现 pyfpgrowth 算法-10 分

直接运行“调用.py”可以得到结果

3.算法应用（20 分）

使用 Apriori 算法在 GroceryStore 数据集上挖掘频繁 3-项集（支持度 0.01）-5 分

一共有 32 个

```
[23]: # 打印三项集
length = 0
for i in B.ans_C:
    i = strtoist(i)
    if len(i)>=3:
        print(i)
        length+=1
print("三项集一共有{}个".format(length))
```

```
[1, 6, 8]
[1, 8, 12]
[1, 12, 43]
[5, 6, 8]
[5, 6, 12]
[5, 8, 12]
[5, 8, 18]
[5, 8, 43]
[5, 12, 43]
[6, 8, 12]
[6, 8, 18]
[6, 8, 27]
[6, 8, 32]
[6, 8, 43]
[6, 8, 61]
[6, 12, 18]
[6, 12, 43]
[6, 12, 61]
[8, 9, 12]
[8, 12, 15]
[8, 12, 18]
[8, 12, 25]
[8, 12, 32]
[8, 12, 35]
[8, 12, 40]
[8, 12, 43]
[8, 12, 51]
[8, 12, 59]
[8, 12, 61]
[8, 12, 66]
[8, 18, 43]
[12, 18, 43]
三项集一共有32个
```

使用 Apriori 算法在 GroceryStore 上挖掘关联规则（支持度 0.01，置信度 0.5）-5 分

```
[26]: B.set_relateRules(0.5)
print("支持度0.01，置信度大于0.5的规则有{}个".format(len(B.relateRules)))
B.relateRules
```

```
支持度0.01，置信度大于0.5的规则有15个
[26]: {'[1, 43]->12': 0.5862068965517241,
 '[5, 6]->8': 0.517361111111111,
 '[5, 43]->8': 0.570048309178744,
 '[5, 43]->12': 0.5845410628019324,
 '[6, 12]->8': 0.5128805620608898,
 '[6, 27]->8': 0.5823529411764706,
 '[6, 43]->8': 0.562992125984252,
 '[6, 61]->8': 0.5245098039215685,
 '[6, 43]->12': 0.5,
 '[9, 12]->8': 0.5175097276264592,
 '[12, 15]->8': 0.5736040609137055,
 '[12, 61]->8': 0.5070422535211268,
 '[12, 66]->8': 0.5525114155251142,
 '[18, 43]->8': 0.5230125523012552,
 '[18, 43]->12': 0.502092050209205}
```

使用 FP-Growth 算法在 GroceryStore 数据集上挖掘的一些关联规则（置信度 0.5）-5 分

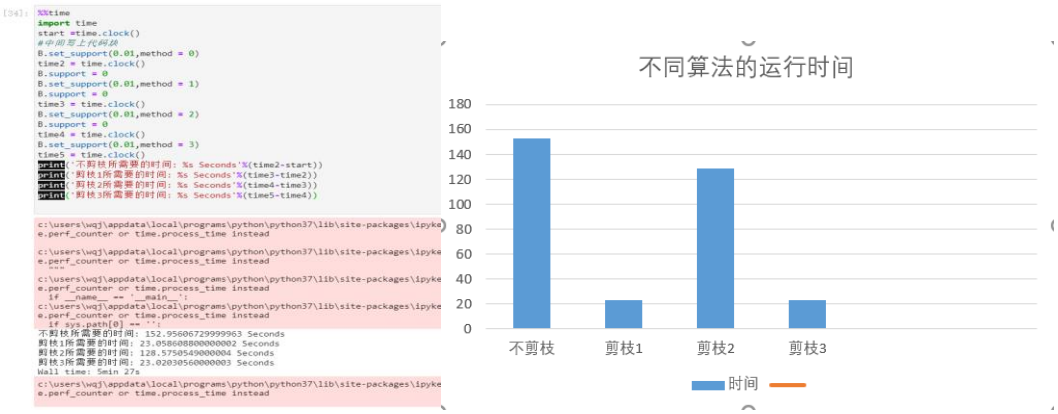
调用.py > ...

```python
import pyfpgrowth
from Data import Data
A = Data("./第二次实践作业数据集/GroceryStore/Groceries.csv")
data = A.data
# 设置支持度和置信度
minS = 0.01
minC = 0.5
# 计算支持值
support = minS*len(data)
# 获取符合支持度规则数据
patterns = pyfpgrowth.find_frequent_patterns(data, support)
# 获取符合置信度规则数据
rules = pyfpgrowth.generate_association_rules(patterns, minC)
print(len(data))
print("获取符合支持度规则数据", patterns)
print("获取符合置信度规则数据", rules)
print(len(patterns))
print(len(rules))

```

```
获取符合置信度规则数据 {(6, 27): ((8,), 0.5823529411764706), (12, 15): ((8,), 0.5736040609137056), (12, 66): ((8,), 0.5525114155251142), (12, 61): ((8,), 0.5070422535211268), (9, 12): ((8,), 0.5175097276264592), (1, 43): ((12,), 0.5862068965517241), (6, 12): ((8,), 0.5128805620608899)}
287
```

算法效率分析，对比 dummyApriori、仅使用第一种剪枝策略 AdvancedApriori_1、同时使用第一种和第二种剪枝策略 AdvancedApriori_12 的时间损耗，画图对比在支持度为 0.01 时，随着频繁项集项的增加，总耗时的变化情况。-5 分



七．实验额外要求（共 30 分，作业给分上限 100 分）

使用命令行接收参数（数据集、支持率、挖掘频繁项集的大小）-5 分

```
PS D:\Desktop\第二次实践> python.exe main.py "./第二次实践作业数据集/GroceryStore/Groceries.csv" 0.01 2
[1, 5]
[1, 6]
[1, 8]
[1, 9]
[1, 12]
[1, 18]
[1, 25]
[1, 32]
[1, 35]
[1, 43]
[1, 51]
[1, 61]
[1, 66]
[3, 6]
[3, 8]
[3, 12]
```

寻找 2 个关联规则，比较强的关联规则，进行市场分析。-5 分

```
'[8, 12]': 0.07483477376715811,
'[8, 18]': 0.05663446873411286,
```

排名靠前的两个关联规则为[8,12],[8,18]通过 json 文件可以知道

"whole milk": 8, "other vegetables": 12, "rolls/buns": 18,

牛奶和蔬菜作为食物经常一起出现

牛奶和面包作为早餐经常一起出现。说明喜欢大家观念中的饮食习惯，一般选择牛奶蔬菜，或者牛奶面包，但是大家选择牛奶蔬菜更多。建议营销健康饮食的概念!!! 迎合大家的胃口

实现第四部分第三种剪枝策略，并将三种剪枝策略一起使用算法运行时间一起画在算法时间对比图表中。-10 分