案例一：基于Apriori挖掘超市商品之间关联性

1 案例目的

• 理解关联规则分析问题的场景；

• 掌握Apriori算法。

2 案例内容

某网上超市收集了1周内所有的交易中购物篮的商品信息。现要根据这些交易数据建立模型，预测顾客买了A商品后还倾向于买哪个B商品。

3 案例知识点

• Python语言编程

• 关联规则分析问题的场景

• Apriori算法

4 案例时长

共2学时，具体安排如下：

• 数据预处理（0.5学时）

• 做关联规则分析（1学时）

• 查看分析结果（0.5学时）

• 增加结果可读性（0.5学时）

5 案例实验环境

**•操作系统：**

1）Linux Ubuntu 16.04 64位操作系统

**•软件环境：**

1）Python 3.7

2）apyori 1.1

**•开发环境与工具：**

1）Spyder 4

6 案例分析

本案例主要分为以下4部分：

1）数据预处理，把数据处理成apriori模型需要的格式；

2）建立关联规则分析模型，挖掘商品之间的关联性。通过设置不同的模型参数检验模型的性能；

3）查看分析结果，理解分析结果的输出格式；

4）处理模型输出的结果，增加结果的可读性。

7 案例实验过程

基于Apriori分析超市商品的相关性，可分为以下6个步骤：

1、导入库；

2、导入数据集；

3、数据预处理；

4、做关联规则分析；

5、查看分析结果；

6、增加结果可读性。

7.1开发准备

在桌面上新建一个目录，名字叫market-goods。本案例中所有的数据集和代码都放在该目录下。

7.1.1 打开Spyder软件

在桌面上点击Spyder图标打开Spyder，并把工作目录切换至market-goods目录下。

7.1.2 获取数据

本案例需要1个数据集，是Market\_Basket\_Optimisation.csv，它是超市购物篮数据。

点击右侧菜单栏中的【资源下载】按钮，复制提供的URL前缀，如https://xxx.eec-cn.com/。

将URL前缀和数据集地址拼接到一起，如https://xxx.eec-cn.com/dataSet/systemLib/733a31f28b6b42b294cc30de7f6b6db0.csv下载数据集。将下载后的数据集Market\_Basket\_Optimisation.csv复制到market-goods目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为market-goods。使用Spyder新建一个Python文件，命名为apriori.py。

至此，整个案例的目录结构如图1所示。



图1 案例的目录结构

7.1.4 了解数据

数据内容如图2所示。

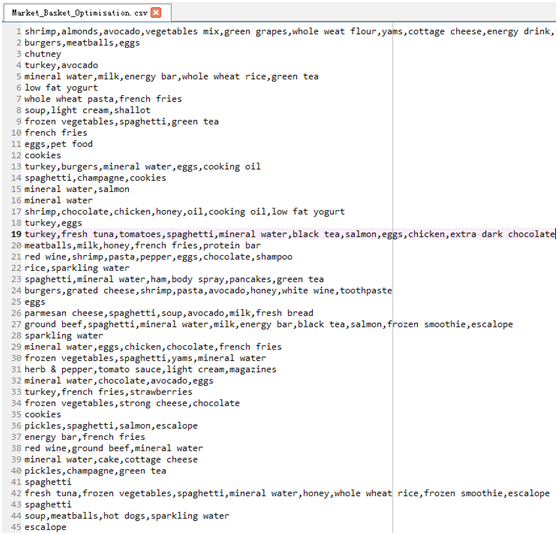


图2 数据样式

数据源存储着7501条交易数据。每一条交易数据都记录了所购买的商品的名称。

通过第2行可以看出，这位顾客买了burgers (肉三明治), meatballs (肉丸), eggs (鸡蛋)；通过第3行数据可以看出，这位顾客只买了chutney (酸辣酱)；通过第33行数据可以看出，顾客买了turkey (火鸡), french fries (炸薯条), strawberries (草莓)。

商品之间是否有关联性呢？请看图3。

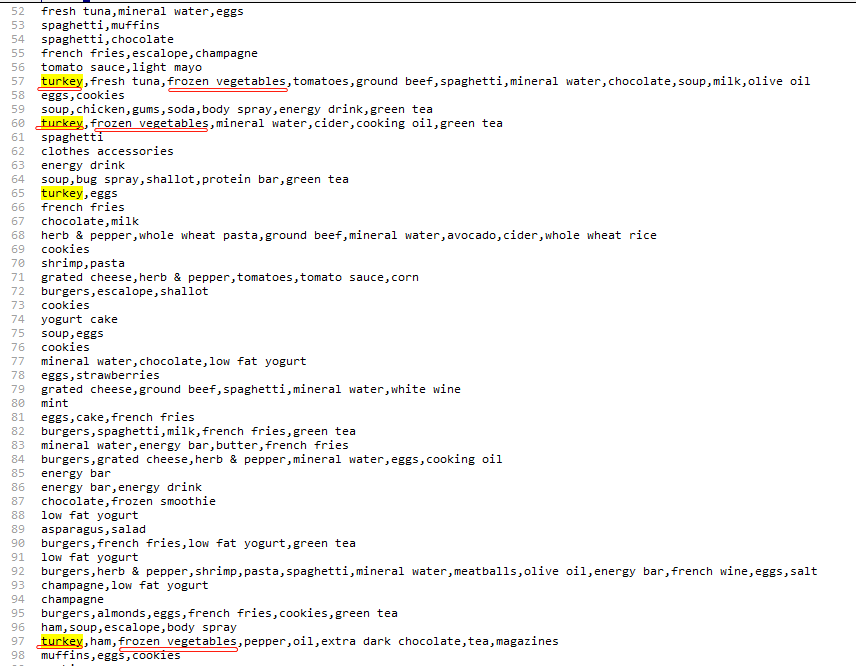


图3 商品之间是否有关联性

通过第57、60、97行数据可以发现，turkey (火鸡)和frozen vegetables (速冻蔬菜)同时出现了好几次。是否意味着买了turkey一般就会买frozen vegetables呢，或者说买了frozen vegetables一般就会买turkey呢？

在用科学的方法分析之前，首先在网上看看turkey和frozen vegetable是否有关联。打开bing搜索（www.bing.com），选择国际版，输入关键词’turkey frozen vegetable’，结果如图4所示。

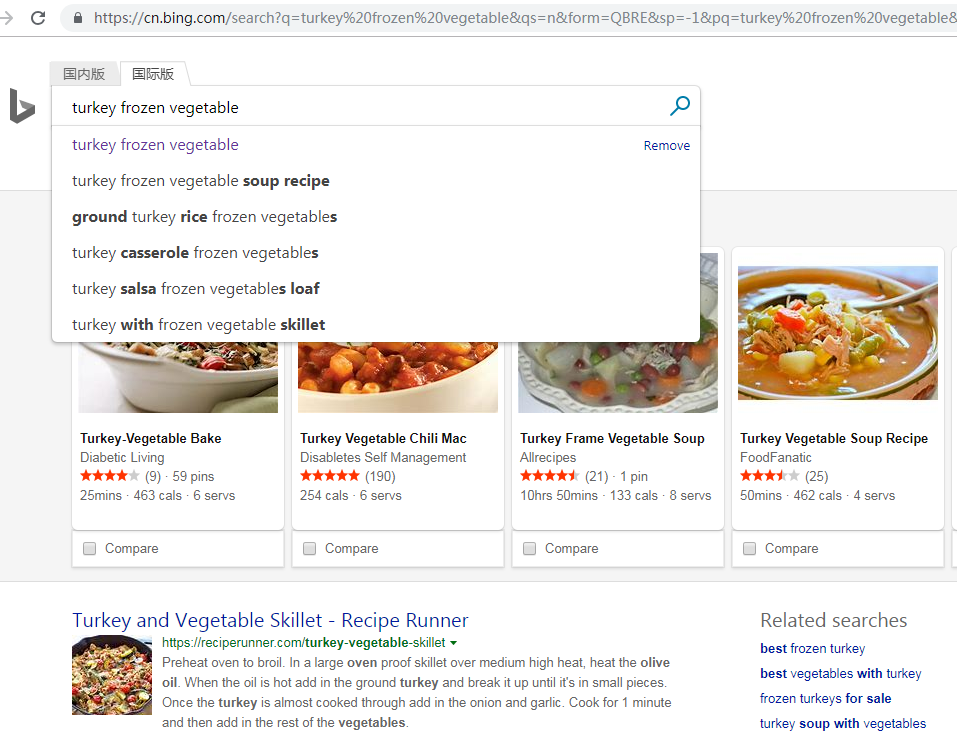


图4 上网搜索turkey和frozen vegetable是否有关系

可以看出，在欧美很多人用turkey和frozen vegetable做汤、炖菜等。所以看起来turkey和frozen vegetable可能有关联。下面用科学的方法来找出所有相关联的商品。

7.2导入库

导入pandas库和apriori库。

【代码7-2】apriori.py

#%%

# 导入包

import pandas as pd

from apyori import apriori

#%%

执行代码块。

7.3导入数据集

导入数据集，查看数据格式。

【代码7-3】apriori.py

#%%

# 导入数据

dataset = pd.read\_csv('Market\_Basket\_Optimisation.csv', header=None)

#%%

执行代码块。

由于这个csv文件没有表头，所以header参数的值为None。

将Spyder的面板切换到Variable explorer，便于随时查看变量的信息。

dataset的变量信息如图5所示。

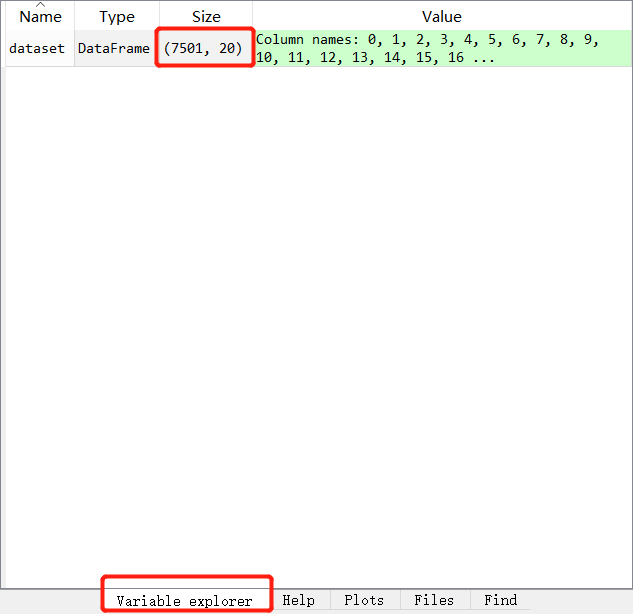


图5 dataset的变量信息

由上图可知，数据集有7501行，20列。

dataset变量的值如图6所示。

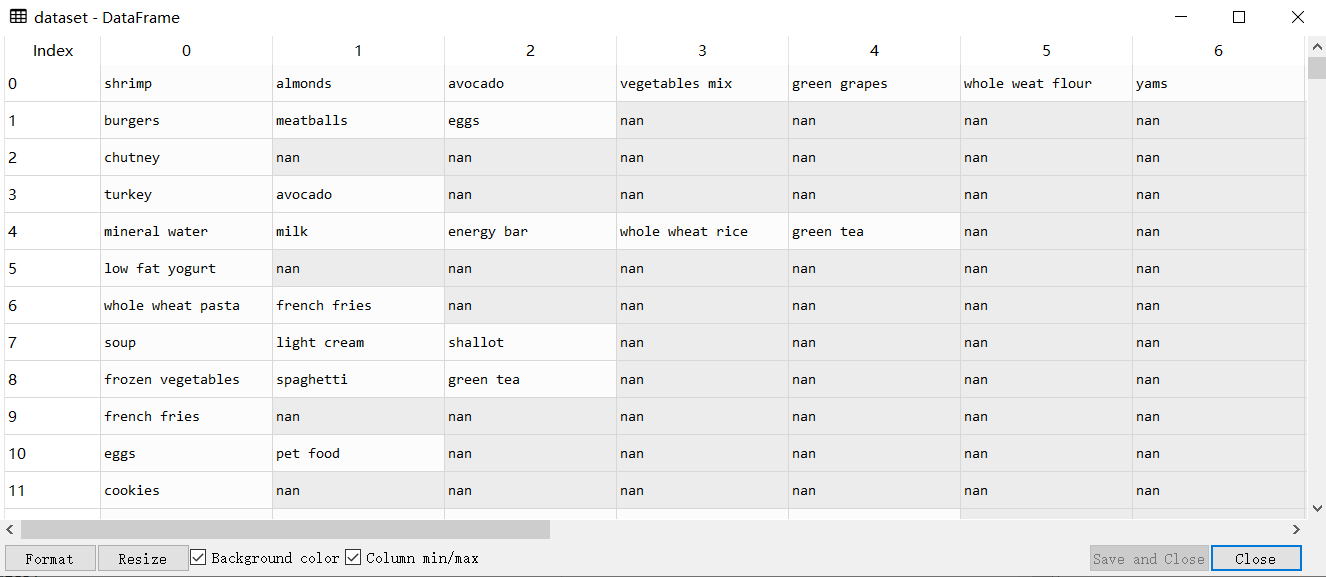


图6 dataset变量的值

由上图可知，数据集有20列，表明在7501条交易记录中，最多的一次买了20个商品。当交易记录不足20时，用nan替代。上图的第2行数据，只买了3个商品，所以剩下的17列数据都是nan。

7.4数据预处理

数据预处理是把数据集中的数据格式处理成apriori所需要的格式。

apriori所需要的数据格式需满足3个条件：

• 数据存储在list中（称之为list A）。list的每一个元素是一条交易记录。

• list A中的每一个元素也是list类型（称之为list B），list B的每一个元素是一个商品。

• 把nan也看做一种商品。即所有的list B的长度都是20（数据集有20列）。

如果上述表达不好理解，请执行代码查看效果，以辅助理解。

【代码7-4】apriori.py

#%%

# 数据预处理

transactions = []

for i in range(0, 7501):

transactions.append([str(dataset.values[i, j]) for j in range(0, 20)])

#%%

执行代码块。

transactions变量的值如图7所示。

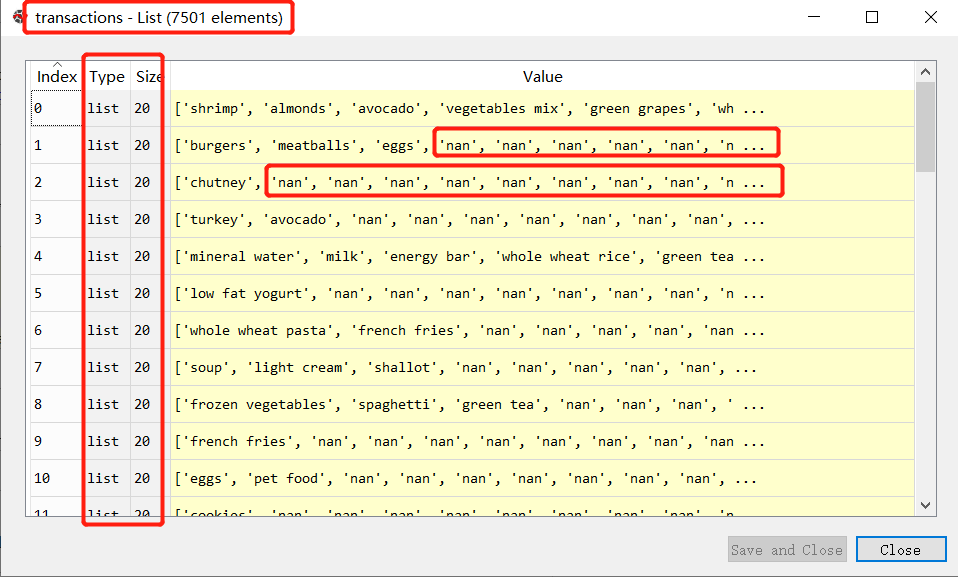


图7 transactions变量的值

7.5做关联规则分析

apriori函数需要传6个参数。

• transactions：传入的数据。数据一定要符合特定的格式。

• min\_support：这个参数的值由业务场景决定。这是一周的交易数据，所以认为平均一天被购买3次，一周被购买7天的商品才有分析的价值。最小support = 3\*7/7501 = 0.0027996。把最小support设为0.003。

• min\_confidence：这个值是试探出来的。在R语言中，这个参数的默认值是0.8。将0.8传入发现apriori函数输出的结果是空。把参数值改为0.4，发现apriori函数输出的结果依然是空。把参数值改为0.2，发现apriori函数输出的结果不再是空。所以把值设为0.2。

• min\_lift：根据apriori的原理，lift指标是最重要的指标。根据经验，lift的值至少是3才能说明商品具有关联性。

• min\_length：这个参数的值由业务场景决定。在本场景中，只关心买了A商品还可能买哪个B商品，所以min\_length和max\_length的值都为2。如果把场景改为：买了A商品和B商品，还可能买哪个C商品，此时min\_length和max\_length的值都为3。

• max\_length：同上。

【代码7-5】apriori.py

#%%

# 做关联规则分析

rules = apriori(transactions = transactions, min\_support = 0.003, min\_confidence = 0.2, min\_lift = 3, min\_length=2, max\_length=2)

#%%

执行代码块。

7.6查看分析结果

使用apriori构建关联规则分析模型后，查看分析结果。

【代码7-6】apriori.py

#%%

# 查看分析结果

results = list(rules)

print(results[0])

#%%

执行代码块。

results[0]的输出结果如图8所示。



图8 results[0]的输出结果

注：results[0]变量的类型是RelationRecord，Spyder的Variable explorer对其支持不友好，所以才通过print()函数查看。绝大多数情况下都是通过Spyder的Variable explorer面板查看变量信息。

由上图可见，light cream（低脂奶油）和chicken（鸡肉）有关联性，lift的值是4.84，confidence的值是0.29，support的值是0.004。

light cream和chicken是有先后顺序的。在图8中，items\_base是light cream，items\_add是chicken，说明顾客先买了light cream后更倾向于再买chicken，并不能说明顾客先买了chicken后更倾向于再买light cream。

下面需要验证light cream和chicken是否真的有很强的关联性。打开bing搜索（www.bing.com），选择国际版，输入关键词’light cream chicken’，结果如图9所示。

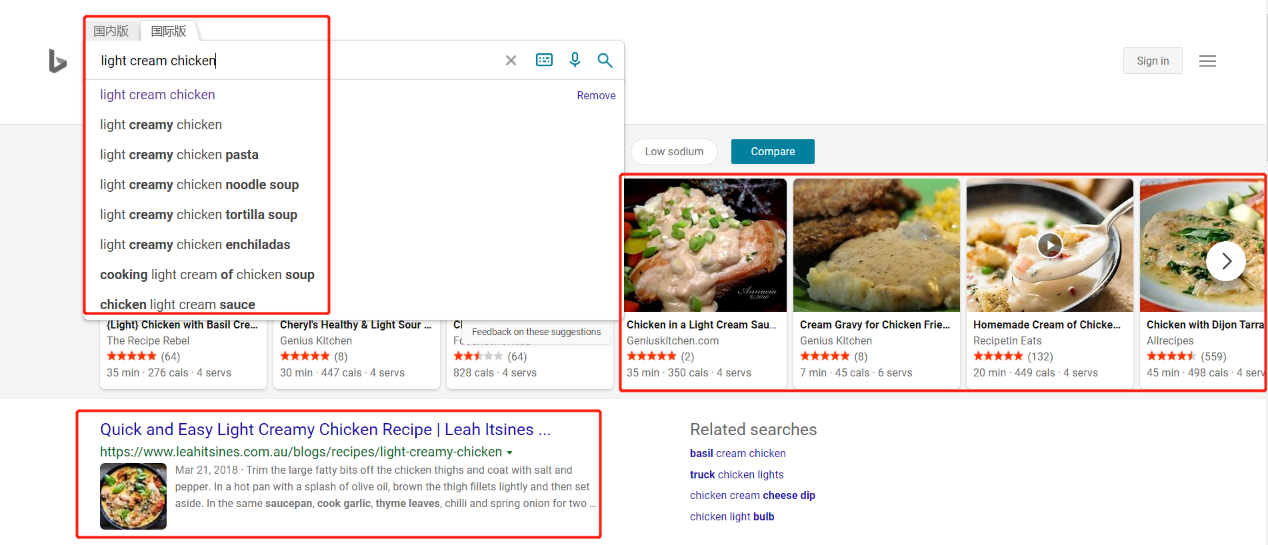


图9 light cream chicken在网上的搜索结果

由上图可知，light cream和chicken可以搭配做很多好吃的，说明他们的关联性很强。

results[0]的可读性并不是很好。下面需要把results的结果存储到DataFrame中，增加结果可读性。

7.7增加结果可读性

为了增加结果可读性，现对results结果做进一步处理。

为了增加结果可读性，首先要了解关联规则分析结果（results[0]）的输出格式。关联规则分析结果（results[0]）已在图8展示。它是RelationRecord类型，里面有3个项，分别为items，support，ordered\_statistics。下面代码分别得到这3项的值。

【代码7-7-1】apriori.py

#%%

# 探索关联规则分析结果的格式-1

result = results[0]

result\_0 = result[0]

result\_1 = result[1]

result\_2 = result[2]

print('result\_0 is: \n', result\_0)

print('result\_1 is: \n', result\_1)

print('result\_2 is: \n', result\_2)

#%%

执行代码块。

result\_0、result\_1、result\_2的结果如图10所示。

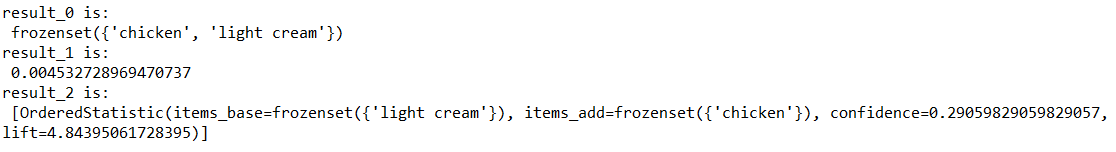


图10 result\_0、result\_1、result\_2的结果

RelationRecord里面有3个项，items是存在关联的对象集合，support存储着support的值，ordered\_statistics存储着其他有意义的数据。

下面主要从ordered\_statistics这一项（result\_2）中提取信息。result\_2是list类型，表明存在一种可能性，items\_base是light cream，items\_add是chicken，同时items\_base是chicken，items\_add是light cream（此时list类型有2个元素）。当apriori函数参数的min\_length和max\_length的值都设为3时，result\_2这个list的元素个数可能更多。在本案例中，light cream和chicken如果有关联性就捆绑销售或者放在同一个货架上，所以谁是items\_base，谁是items\_add已无所谓。所以这里只关心result[2][0]。

下面从result[2][0]中提取items\_base、items\_add、confidence、lift信息，代码如下：

【代码7-7-2】apriori.py

#%%

# 探索关联规则分析结果的格式-2

result\_2\_0 = result[2][0]

print('result\_2\_0 is: \n', result\_2\_0)

result\_2\_0\_0 = result[2][0][0]

result\_2\_0\_1 = result[2][0][1]

result\_2\_0\_2 = result[2][0][2]

result\_2\_0\_3 = result[2][0][3]

print('result\_2\_0\_0 is: \n', result\_2\_0\_0)

print('result\_2\_0\_1 is: \n', result\_2\_0\_1)

print('result\_2\_0\_2 is: \n', result\_2\_0\_2)

print('result\_2\_0\_3 is: \n', result\_2\_0\_3)

#%%

执行代码块。

result\_2\_0、result\_2\_0\_0、result\_2\_0\_1、result\_2\_0\_2、result\_2\_0\_3的结果如图11所示。

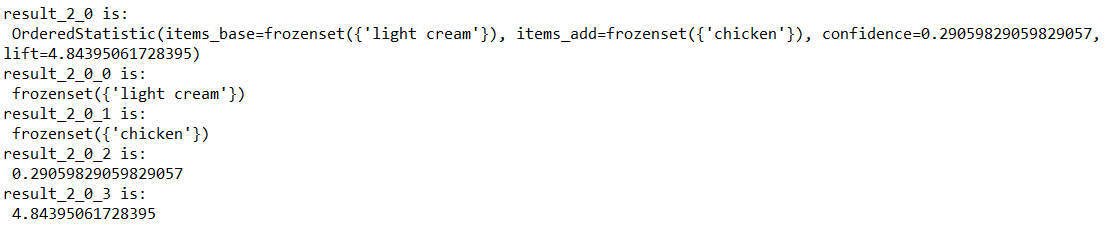


图11 result\_2\_0、result\_2\_0\_0、result\_2\_0\_1、result\_2\_0\_2、result\_2\_0\_3的结果

由上图可见，result\_2\_0\_0对应items\_base的值，result\_2\_0\_1对应items\_add的值，result\_2\_0\_2对应confidence的值，result\_2\_0\_3对应lift的值。

下面写一个函数，把上面的代码整合，即把support的值、items\_base的值、items\_add的值、confidence的值、lift的值提取出来。然后把结果存储到DataFrame中。代码如下：

【代码7-7-3】apriori.py

#%%

# 增加结果可读性

def inspect(results):

lhs = [tuple(result[2][0][0])[0] for result in results]

rhs = [tuple(result[2][0][1])[0] for result in results]

supports = [result[1] for result in results]

confidences = [result[2][0][2] for result in results]

lifts = [result[2][0][3] for result in results]

return list(zip(lhs, rhs, supports, confidences, lifts))

results\_in\_dataframe = pd.DataFrame(inspect(results), columns=['先买的商品','后买的商品','Supports','Confidences','Lifts'])

#%%

执行代码块。

把results\_in\_dataframe的值按lift排序，因为lift的值越大，说明关联性越高。代码如下：

【代码7-7-4】apriori.py

#%%

# 按lift倒序排序

results\_in\_dataframe = results\_in\_dataframe.nlargest(n=10, columns='Lifts')

#%%

执行代码块。

results\_in\_dataframe变量的值如图12所示。

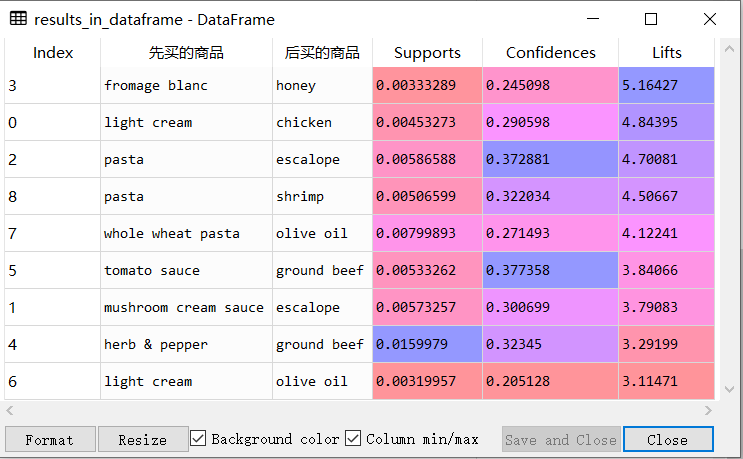


图12 results\_in\_dataframe变量的值

由上图可见，顾客购买了fromage blanc（自制酸奶奶酪），非常倾向于购买honey（蜂蜜）；

顾客购买了pasta（意大利面），非常倾向于购买escalope（肉片）......

结论：使用关联规则分析算法，可以找到有关联性的商品。

8 案例代码

【案例代码】apriori.py

'''

基于Apriori分析超市商品的相关性

安装apyori使用如下命令： pip install apyori

'''

#%%

# 导入包

import pandas as pd

from apyori import apriori

#%%

#%%

# 导入数据

dataset = pd.read\_csv('Market\_Basket\_Optimisation.csv', header=None)

#%%

#%%

# 数据预处理

transactions = []

for i in range(0, 7501):

transactions.append([str(dataset.values[i, j]) for j in range(0, 20)])

#%%

#%%

# 做关联规则分析

rules = apriori(transactions = transactions, min\_support = 0.003, min\_confidence = 0.2, min\_lift = 3, min\_length=2, max\_length=2)

#%%

#%%

# 查看分析结果

results = list(rules)

print(results[0])

#%%

#%%

# 探索关联规则分析结果的格式-1

result = results[0]

result\_0 = result[0]

result\_1 = result[1]

result\_2 = result[2]

print('result\_0 is: \n', result\_0)

print('result\_1 is: \n', result\_1)

print('result\_2 is: \n', result\_2)

#%%

#%%

# 探索关联规则分析结果的格式-2

result\_2\_0 = result[2][0]

print('result\_2\_0 is: \n', result\_2\_0)

result\_2\_0\_0 = result[2][0][0]

result\_2\_0\_1 = result[2][0][1]

result\_2\_0\_2 = result[2][0][2]

result\_2\_0\_3 = result[2][0][3]

print('result\_2\_0\_0 is: \n', result\_2\_0\_0)

print('result\_2\_0\_1 is: \n', result\_2\_0\_1)

print('result\_2\_0\_2 is: \n', result\_2\_0\_2)

print('result\_2\_0\_3 is: \n', result\_2\_0\_3)

#%%

#%%

# 增加结果可读性

def inspect(results):

lhs = [tuple(result[2][0][0])[0] for result in results]

rhs = [tuple(result[2][0][1])[0] for result in results]

supports = [result[1] for result in results]

confidences = [result[2][0][2] for result in results]

lifts = [result[2][0][3] for result in results]

return list(zip(lhs, rhs, supports, confidences, lifts))

results\_in\_dataframe = pd.DataFrame(inspect(results), columns=['先买的商品','后买的商品','Supports','Confidences','Lifts'])

#%%

#%%

# 按lift倒序排序

results\_in\_dataframe = results\_in\_dataframe.nlargest(n=10, columns='Lifts')

#%%