In [94]:
```
# -*- coding: utf-8 -*-
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from pandas import DataFrame,Series
from scipy import stats
import orangecontrib.associate.fpgrowth as oaf
from pymining import itemmining, assocrules
import fp_growth_py3 as fpg
```

一.数据读入

首先读入数据集winemag-data-130k-v2.csv

In [95]:
```
filename='C:/Users/Administrator/Desktop/coda/wine-reviews/winemag-data-130k-v2.csv'
df=pd.read_csv(filename)
```

数据集中共有129971条记录，有14个属性，我们就要对这些数据进行关联分析

In [96]:
```
cols = df.columns
print(df.shape)
print(cols)
print(df.iloc[:,0].size)
```

```
(129971, 14)
Index(['Unnamed: 0', 'country', 'description', 'designation', 'points',
       'price', 'province', 'region_1', 'region_2', 'taster_name',
       'taster_twitter_handle', 'title', 'variety', 'winery'],
      dtype='object')
129971
```

二.数据处理

数据中一共有'country', 'description', 'designation', 'points','price', 'province', 'region_1', 'region_2', 'taster_name','taster_twitter_handle', 'title', 'variety', 'winery'这13个属性可供我们分析，我们将没有关联关系的评价属性（description）给去掉后剩下12个属性。

其中points和price为数值属性，我们将其离散化处理，将其转化为字符串：

对于points，其数值范围为80到100，我们每5分为一个阶段我们将其分为80-85，85-90，90-95，95-100这四个阶段。

对于price，根据上次我们的统计结果，其分布范围较大，最小为4美元，最大为2000美元，但是其主要的值都分布在100美元以内，所以我们根据第一次作业的直方图统计结果将其分为0-16，16-25，25-50，50-100，100+这五个阶段。

有了数据之后，我们先要对原始数据进行编码。

我们需要将数据放入到一个列表中用来分析，每个列表的一行代表一个数据输出的项目，设该列表名字是listToAnalysis，

则生成该列表的代码如下：

In [97]:
```
lists=[]
listToAnalysis = []
for i in range(df.iloc[:,0].size):
    s=df.iloc[i]['country']
    s='country:'+str(s)
    lists.append(s)
    s=df.iloc[i]['points']
    if s<85:
        s='points'+'80_85'
    elif s<90:
        s='points'+'85_90'
    elif s<95:
        s='points'+'90_95'
    else:
        s='points'+'95_100'
    lists.append(s)
    s=df.iloc[i]['price']
    if s>=0 and s<=16:
        s='price'+'0_16'
    elif  s<=25:
        s='price'+'16_25'
    elif s<=50:
        s='price'+'25_50'
    elif s<=100:
        s='price'+'50_100'
    elif s>100:
        s='prince'+'100_'
    else:
        s='price nan'
    lists.append(s)
    s=df.iloc[i]['province']
```

```
    s='province:'+str(s)
    lists.append(s)
    s=df.iloc[i]['designation']
    s='designation:'+str(s)
    lists.append(s)
    s=df.iloc[i]['region_1']
    s='region_1:'+str(s)
    lists.append(s)
    s=df.iloc[i]['region_2']
    s='region_2:'+str(s)
    lists.append(s)
    s=df.iloc[i]['taster_name']
    s='taster_name:'+str(s)
    lists.append(s)
    s=df.iloc[i]['taster_twitter_handle']
    s='taster_twitter_handle:'+str(s)
    lists.append(s)
    s=df.iloc[i]['title']
    s='title:'+str(s)
    lists.append(s)
    s=df.iloc[i]['variety']
    s='variety:'+str(s)
    lists.append(s)
    s=df.iloc[i]['winery']
    s='winery:'+str(s)
    lists.append(s)
    listToAnalysis.append(lists.copy())
    lists.clear()
```

我们将生成的列表的前1000项输出展示给大家：

In [5]:
```
for i in range(1000):
    print(listToAnalysis[i])
```

```
na Lee C. Iijima', 'taster_twitter_handle:nan', 'title:Schmitt Söhne 2015 Riesling (Rheinhessen)', 'variety:Riesling', 'winery:Schmitt Söhne']
['country:Australia', 'points85_90', 'price16_25', 'province:South Australia', 'designation:Made With Organic Grapes', 'region_1:South Australia', 'region_2:nan', 'taster_name:Joe Czerwinski', 'taster_twitter_handle:@JoeCz', 'title:Yalumba 2016 Made With Organic Grapes Chardonnay (South Australia)', 'variety:Chardonnay', 'winery:Yalumba']
['country:US', 'points85_90', 'price16_25', 'province:Oregon', 'designation:Rosé of', 'region_1:Eola-Amity Hills', 'region_2:Willamette Valley', 'taster_name:Paul Gregutt', 'taster_twitter_handle:@paulgwine\xa0', "title:Z'IVO 2015 Rosé of Pinot Noir (Eola-Amity Hills)", 'variety:Pinot Noir', 'winery:Z'IVO"]
['country:Portugal', 'points85_90', 'price nan', 'province:Tejo', 'designation:Bridão', 'region_1:nan', 'region_2:nan', 'taster_name:Roger Voss', 'taster_twitter_handle:@vossroger', 'title:Adega Cooperativa do Cartaxo 2014 Bridão Touriga Nacional (Tejo)', 'variety:Touriga Nacional', 'winery:Adega Cooperativa do Cartaxo']
['country:Chile', 'points85_90', 'price0_16', 'province:Rapel Valley', 'designation:Special Release Reserva', 'region_1:nan', 'region_2:nan', 'taster_name:Michael Schachner', 'taster_twitter_handle:@wineschach', 'title:Aresti 2014 Special Release Reserva Carmenère (Rapel Valley)', 'variety:Carmenère', 'winery:Aresti']
['country:Spain', 'points85_90', 'price0_16', 'province:Galicia', 'designation:nan', 'region_1:Rías Baixas', 'region_2:nan', 'taster_name:Michael Schachner', 'taster_twitter_handle:@wineschach', 'title:Spyro 2014 Albariño (Rías Baixas)', 'variety:Albariño', 'winery:Spyro']
['country:France', 'points85_90', 'price0_16', 'province:France Other', 'designation:La Réserve', 'region_1:Vin de France', 'region_2:nan', 'taster_name:Roger Voss', 'taster_twitter_handle:@vossroger', 'title:Lionel Osmin & Cie 2016 La Réserve Petit Manseng (Vin de France)', 'variety:Petit Manseng', 'winery:Lionel Osmin & Cie']
['country:Australia', 'points85_90', 'price16_25', 'province:South Australia', 'designation:Jester Sangiovese', 'region_1:McLaren Vale',
```

三.找频繁项集

将输入数据的格式完成之后，就可以使用关联规则函数进行数据挖掘。
关联分析属于数据挖掘的一大类。计算频繁项集有两种方法，分别是Apriori算法和FP-growth算法，但是，FP-growth算法比Apriori算法时间复杂度低。我们分别用两种方法对本数据集进行了关联分析，得到了相同的结构。

Apriori算法使用一种称为逐层搜索的迭代方法，其中k项集用于探索（k+1）项集。首先，通过扫描数据库，累计每个项的计数，并收集满足最小支持度的项，找出频繁1项集的集合，记为L1。然后，使用L1找出频繁2项集的集合L2，使用L2找出L3，如此下去，直到不能再找到频繁k项集。

3.1 FP-growth算法

FP-growth算法是伊利罗伊香槟分校的韩嘉炜教授于2004年提出的，它是为了解决Apriori算法每次增加频繁项集的大小都要遍历整个数据库的缺点，特别是当数据集很大时，该算法执行速度要快于Apriori算法两个数量级。FP-growth算法的任务是将数据集存储在一个特定的称为FP树的结构之后发现频繁项集或者频繁项对，虽然它能够高效地发现频繁项集，但是不能用来发现关联规则，也就是只优化了Apriori算法两个功能中的前一个功能。

FP-growth算法只需要对数据集进行两次扫描，所以即使数据集很大时也不会花费太多的时间在扫描数据上，它发现频繁项集的基本过程如下：

1）构建FP树

2）从FP树中挖掘频繁项集

In [98]:
```
frequent_itemsets = fpg.find_frequent_itemsets(listToAnalysis, minimum_support=100, include_support=True)
```

In [99]:
```
result = []
for itemset, support in frequent_itemsets:    # 将generator结果存入list
    result.append((itemset, support))
```

In [100]:
```
result = sorted(result, key=lambda i: i[0])   # 排序后输出
```

我们将在数据集中出现次数大于100的频繁项集使用FP-growth方法找出来，共计137043项

我们将频繁项集以及其出现的次数经过排序后输出如下：

In [20]:
```
for itemset, support in result:
    print(str(itemset) + ' ' + str(support))
```

```
['region_2:nan', 'points85_90', 'region_1:nan', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 158
['designation:nan', 'region_1:nan', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 145
['region_2:nan', 'designation:nan', 'region_1:nan', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 145
['region_2:nan', 'region_1:nan', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 395
['points90_95', 'region_1:nan', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 220
['region_2:nan', 'points90_95', 'region_1:nan', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 220
['price0_16', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 108
['region_2:nan', 'price0_16', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 108
['points85_90', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 158
['region_2:nan', 'points85_90', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 158
['price16_25', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 134
['region_2:nan', 'price16_25', 'taster_name:Anne Krebiehl\xaOMW', 'country:Austria', 'province:Burgenland'] 134
['points90_95', 'country:Austria', 'province:Burgenland'] 322
['region_2:nan', 'points90_95', 'country:Austria', 'province:Burgenland'] 322
['taster_name:Roger Voss', 'country:Austria', 'province:Burgenland'] 277
['region_2:nan', 'taster_name:Roger Voss', 'country:Austria', 'province:Burgenland'] 277
['points85_90', 'taster_name:Roger Voss', 'country:Austria', 'province:Burgenland'] 157
['region_2:nan', 'points85_90', 'taster_name:Roger Voss', 'country:Austria', 'province:Burgenland'] 157
['points90_95', 'taster_name:Roger Voss', 'country:Austria', 'province:Burgenland'] 102
['region_2:nan', 'points90_95', 'taster_name:Roger Voss', 'country:Austria', 'province:Burgenland'] 102
```

3.2 Apriori算法

如果某个项集是频繁的，那么他的子集也是频繁的。例如，有四种商品，0,1,2,3，其中{0,1}是频繁项集，即同时购买商品0和1的购买行为数量在总的购买行为占比较大（一般会有认为给定的最小支持度的阈值来区分是否是频繁项集），那么，{0}和{1}也是频繁项集。反之，如果如果项集不是频繁项集，那么他的超集也不是频繁项集，例如，如果{0}不是频繁项集，那么{0,1}，{0,1,2}…也不是频繁项集。

Apriori算法使用一种称为逐层搜索的迭代方法，其中k项集用于探索（k+1）项集。首先，通过扫描数据库，累计每个项的计数，并收集满足最小支持度的项，找出频繁1项集的集合，记为L1。然后，使用L1找出频繁2项集的集合L2，使用L2找出L3，如此下去，直到不能再找到频繁k项集。

Apriori计算频繁项集的命令如下：

In [87]:
```
relim_input = itemmining.get_relim_input(listToAnalysis)
```

In [88]:
```
report = itemmining.relim(relim_input, min_support=100)
```

In [72]:
```
print(len(report))
sum=len(report)
```

```
137043
```

四.关联规则计算以及评价标准

4.1 关联规则及其支持度和置信度计算

1.关联规则的一般形式
项集A、B同时发生的概率称为关联规则的支持度（也称相对支持度）
support(A->B)=P(A并B)
项集A发生，则项集B发生的概率为关联规则的置信度。
confidence(A->B)=P(A|B)

2.最小支持度和最小置信度
最小支持度是用户或专家定义的衡量支持度的一个阈值，表示项目集在统计意义上的最低重要性，最小置信度是用户或专家定义的衡量置信度的阈值，表示关联规则的最低重要性，最小置信度是用户或专家定义的衡量置信度的一个阈值，表示关联规则的最低可靠性。同事满足最小支持度阈值和最小置信度阈值的规则称作强规则。

4.2 关联规则评价方法

4.2.1 相关性系数Lift

lift(A,B)=P(A交B)/(P(A)*P(B))

如果lift(A,B)>1表示A、B呈正相关，lift(A,B)<1表示A、B呈负相关，lift(A,B)=1表示A、B不相关（独立）。
实际运用中，正相关和负相关都是我们需要关注的，而独立往往是我们不需要的，lift(A,B)等于1的情形也很少，一般只要接近于1我们就认为是独立了。

4.2.2 全自信度

全自信度all_confidence的定义如下：

all_confidence(A,B)=P(A交B)/max{P(A),P(B)}

=min{P(B|A),P(A|B)}

=min{confidence(A－>B),confidence(B－>A)}

4.2.3 最大自信度

最大自信度则与全自信度相反，求的不是最小的支持度而是最大的支持度，
max_confidence(A,B)=max{confidence(A－>B),confidence(B－>A)}，
不过感觉最大自信度不太实用。

4.2 4 Kulc

Kulc系数就是对两个自信度做一个平均处理：
kulc(A,B)=(confidence(A－>B)+confidence(B－>A))/2。
kulc系数是一个很好的度量标准。
该公式表示 将两种事件作为条件的置信度的均值,避开了支持度的计算,因此不会受零和事务的影响。

4.2.5 cosine(A,B)

cosine(A,B)=P(A交B)/sqrt(P(A)*P(B))
=sqrt(P(A|B)*P(B|A))
=sqrt(confidence(A－>B)*confidence(B－>A))

评价结构分析

因为数据集数据量过大，而且是多个地区的统计数据混合而成的，所以我们再用支持度来衡量是否频繁就不太合适（假如12w条数据中，美国的只有100条，那么我们如果将支持度设为0.1那么美国的数据都会被我们忽视），所以我们用出现的次数作为衡量标准，我们将最少出现次数度为100,最小支持度为0.5的关联规则输出出来，

In [89]:
```
rules1 = assocrules.mine_assoc_rules(report, min_support=2, min_confidence=0.5)
```

In [90]:
```
import math
```

In [102]:
```
for a,b,c,d in rules1:
    z=report[a|b]*1.0/sum
    x=report[b]*1.0/sum
    y=report[a]*1.0/sum
    conf1=z/y;
    conf2=z/x;
    all_conf=min(conf1,conf2)
    if (len(a)<=3 and len(b)<=3 and all_conf>0.1):

        lift=z/(x*y)
        max_conf=max(conf1,conf2)
        kulc=(conf1+conf2)/2
        cosine=math.sqrt(conf1*conf2)
        print(a)
        print('-->')
        print(b)
        print('   出现次数：',c)
        print('      支持度：',y)
        print('      置信度：',d)
        print('评价方法：————————————————————')
        print('       Lift值为：',lift)
        print('    全置信度为：',all_conf)
        print('  最大置信度为：',max_conf)
        print('           KULC：',kulc)
        print('       cosine为：',cosine)
        print('————————————————————————————\n')
```

```
frozenset({'variety:Grüner Veltliner', 'points90_95'})
-->
frozenset({'country:Austria', 'region_2:nan'})
   出现次数： 738
      支持度： 0.00594748059182433
      置信度： 0.9547218628719275
评价方法：————————————————————
      Lift值为： 37.09601053492595
    全置信度为： 0.22062780269058296
  最大置信度为： 0.9547218628719275
           KULC： 0.5876748327812552
       cosine为： 0.4589533601425023
————————————————————————————

frozenset({'variety:Grüner Veltliner'})
-->
frozenset({'country:Austria', 'region_2:nan', 'points90_95'})
   出现次数： 738
      支持度： 0.010348462349293304
      置信度： 0.5486988847583643
```

从我们最后的分析结果来看，我们找出了大量的具有关联的规则（比如省份与国家之间的关系以及评分与价格地区之间的联系）它们都在如上的输出框中给予展示，它们的lift的值大多数都大于1，说明它们之间存在着一定的正相关关系。