前言

本文档为针对Wine Review数据集进行频繁模式与关联规则挖掘的报告。
报告整体分为四个模块介绍：

1. 数据集简介
2. 任务分析
3. 具体流程
4. 可视化展示与验证

以下将逐步展开介绍各部分内容以及操作过程。

一. 数据集简介

1. Wine Review是包含了两个数据子集的红酒评论数据集
2. 两个子数据集，分别含有150930和129971条数据
3. 数据集1、2公有属性种类共计11种：

属性名	Unnamed: 0	country	description	designation	points	price	province	region_1	region_2	variety	winery
中文名	序号	国家	口味描述	葡萄酒名称	打分	价格	省份	产地1	产地2	葡萄品种	酒厂
数据类型	整型	字符串	字符串	字符串	整型	浮点数	字符串	字符串	字符串	字符串	字符串

4. 数据集2独有属性3种：

属性名	taster_name	taster_twitter_handle	title
中文名	品鉴师姓名	品鉴师推特	标签
数据类型	字符串	字符串	字符串

二. 任务分析

根据数据集情况，我们选取数据子集2，并将**{points（打分）、price（价格）、region_1（主产地）、variety（葡萄品种）、taster_twitter_handle(品鉴师twitter)、winery（酒厂）}**作为所有项集来挖掘其中的频繁模式，寻找并分析其中的关联规则。

三. 具体流程

0.导入数据集并进行预处理

```
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt
import pylab
%matplotlib inline

#导入数据集
data = pd.read_csv("./winemag-data-130k-v2.csv",encoding="utf-8")

#删除无需项集和所需项集值为空的数据
A=data.drop(['Unnamed:
0','country','description','designation','province','region_2','taster_name','ti
tle'],1).dropna()
display(A.columns)
A.describe()
```

```
Index(['points', 'price', 'region_1', 'taster_twitter_handle', 'variety',
       'winery'],
      dtype='object')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	points	price
count	74292.000000	74292.000000
mean	88.732946	37.408039
std	2.957283	45.347596
min	80.000000	4.000000
25%	87.000000	18.000000
50%	89.000000	28.000000
75%	91.000000	45.000000
max	100.000000	3300.000000

1.将数据转为适合进行关联规则挖掘的形式，构造事务集

- points(打分)和price(价格)为数值数据，根据平均值将其转为标称数据
- 剩余属性在原字符串后加上属性名作为标识，方便之后分析

```
#处理points和price数据
A['points'] = A['points'].apply(lambda x:'high_points' if x > 89.0 else 'low_points' )
A['price'] = A['price'].apply(lambda x:'expensive' if x > 28.0 else 'cheap' )
A['region_1'] = A['region_1'].apply(lambda x:x+'_region')
A['taster_twitter_handle'] = A['taster_twitter_handle'].apply(lambda x:x+'_twitter')
A['variety'] = A['variety'].apply(lambda x:x+'_variety')
A['winery'] = A['winery'].apply(lambda x:x+'_winery')

#构造事务集
transactions = []
for index, row in A.iterrows():
    transactions +=
[(row['points'],row['price'],row['region_1'],row['taster_twitter_handle'],row['variety'],
row['winery'])]

display(transactions[:5])
```

```
[('low_points',
  'cheap',
  'Willamette Valley_region',
  '@paulgwine\xa0_twitter',
  'Pinot Gris_variety',
  'Rainstorm_winery'),
 ('low_points',
  'expensive',
  'Willamette Valley_region',
  '@paulgwine\xa0_twitter',
  'Pinot Noir_variety',
  'Sweet Cheeks_winery'),
 ('low_points',
  'cheap',
  'Navarra_region',
  '@wineschach_twitter',
  'Tempranillo-Merlot_variety',
  'Tandem_winery'),
 ('low_points',
  'cheap',
  'Vittoria_region',
  '@kerinokeefe_twitter',
  'Frappato_variety',
  'Terre di Giurfo_winery'),
 ('low_points',
  'cheap',
  'Alsace_region',
  '@vossroger_twitter',
  'Gewürztraminer_variety',
  'Trimbach_winery')]
```

2. 寻找频繁子项

由于需要分析的属性数据量较大，而apriori搜索效率较低，我们参考使用改进后的efficient-apriori算法，具体如下：

```
import itertools
import numbers
import typing
import collections
from dataclasses import field, dataclass
import collections.abc

@dataclass
class ItemsetCount:
    itemset_count: int = 0
    members: set = field(default_factory=set)


class TransactionManager:
    def __init__(self, transactions: typing.Iterable[typing.Iterable[typing.Hashable]]):
        self._indices_by_item = collections.defaultdict(set)
        i = -1
        for i, transaction in enumerate(transactions):
            for item in transaction:
                self._indices_by_item[item].add(i)

        # Total number of transactions
        self._transactions = i + 1

    @property
    def items(self):
        return set(self._indices_by_item.keys())

    def __len__(self):
        return self._transactions

    def transaction_indices(self, transaction: typing.Iterable[typing.Hashable]):

        transaction = set(transaction)
        item = transaction.pop()
        indices = self._indices_by_item[item]
        while transaction:
            item = transaction.pop()
            indices = indices.intersection(self._indices_by_item[item])
        return indices

    def transaction_indices_sc(self, transaction: typing.Iterable[typing.Hashable],
min_support: float = 0):
        transaction = sorted(transaction, key=lambda item:
len(self._indices_by_item[item]), reverse=True)
        item = transaction.pop()
        indices = self._indices_by_item[item]
        support = len(indices) / len(self)
        if support < min_support:
            return False, None
        while transaction:
            item = transaction.pop()
```

```
            indices = indices.intersection(self._indices_by_item[item])
            support = len(indices) / len(self)
            if support < min_support:
                return False, None
        return True, indices


def join_step(itemsets: typing.List[tuple]):
    i = 0
    while i < len(itemsets):
        skip = 1
        *itemset_first, itemset_last = itemsets[i]
        tail_items = [itemset_last]
        tail_items_append = tail_items.append  # Micro-optimization
        for j in range(i + 1, len(itemsets)):
            *itemset_n_first, itemset_n_last = itemsets[j]
            if itemset_first == itemset_n_first:
                tail_items_append(itemset_n_last)
                skip += 1
            else:
                break
        itemset_first_tuple = tuple(itemset_first)
        for a, b in sorted(itertools.combinations(tail_items, 2)):
            yield itemset_first_tuple + (a,) + (b,)
        i += skip


def prune_step(itemsets: typing.Iterable[tuple], possible_itemsets: typing.List[tuple]):
    itemsets = set(itemsets)

    for possible_itemset in possible_itemsets:
        for i in range(len(possible_itemset) - 2):
            removed = possible_itemset[:i] + possible_itemset[i + 1 :]
            if removed not in itemsets:
                break
        else:
            yield possible_itemset


def apriori_gen(itemsets: typing.List[tuple]):
    possible_extensions = join_step(itemsets)
    yield from prune_step(itemsets, possible_extensions)


def itemsets_from_transactions(
    transactions: typing.Iterable[typing.Union[set, tuple, list]],
    min_support: float,
    max_length: int = 8,
    verbosity: int = 0,
    output_transaction_ids: bool = False,
):
    if not (isinstance(min_support, numbers.Number) and (0 <= min_support <= 1)):
        raise ValueError("`min_support` must be a number between 0 and 1.")

    manager = TransactionManager(transactions)

    transaction_count = len(manager)
    if transaction_count == 0:
```

```
        return dict(), 0  # large_itemsets, num_transactions
    if verbosity > 0:
        print("Generating itemsets.")
        print(" Counting itemsets of length 1.")

    candidates: typing.Dict[tuple, int] = {(item,): len(indices) for item, indices in
manager._indices_by_item.items()}
    large_itemsets: typing.Dict[int, typing.Dict[tuple, int]] = {
        1: {item: count for (item, count) in candidates.items() if (count / len(manager))
>= min_support}
    }

    if verbosity > 0:
        print("  Found {} candidate itemsets of length 1.".format(len(manager.items)))
        print("  Found {} large itemsets of length 1.".format(len(large_itemsets.get(1,
dict()))))
    if verbosity > 1:
        print("    {}".format(list(item for item in large_itemsets.get(1,
dict()).keys())))

    if not large_itemsets.get(1, dict()):
        return dict(), 0  # large_itemsets, num_transactions
    k = 2
    while large_itemsets[k - 1] and (max_length != 1):
        if verbosity > 0:
            print(" Counting itemsets of length {}.".format(k))
        itemsets_list = sorted(item for item in large_itemsets[k - 1].keys())
        C_k: typing.List[tuple] = list(apriori_gen(itemsets_list))

        if verbosity > 0:
            print("  Found {} candidate itemsets of length {}.".format(len(C_k), k))
        if verbosity > 1:
            print("   {}".format(C_k))
        if not C_k:
            break
        if verbosity > 1:
            print("    Iterating over transactions.")

        found_itemsets: typing.Dict[tuple, int] = dict()
        for candidate in C_k:
            over_min_support, indices = manager.transaction_indices_sc(candidate,
min_support=min_support)
            if over_min_support:
                found_itemsets[candidate] = len(indices)
        if not found_itemsets:
            break
        large_itemsets[k] = {i: counts for (i, counts) in found_itemsets.items()}

        if verbosity > 0:
            num_found = len(large_itemsets[k])
            print("  Found {} large itemsets of length {}.".format(num_found, k))
        if verbosity > 1:
            print("   {}".format(list(large_itemsets[k].keys())))
        k += 1
        if k > max_length:
            break

    if verbosity > 0:
```

```
        print("Itemset generation terminated.\n")

    if output_transaction_ids:
        itemsets_out = {
            length: {
                item: ItemsetCount(itemset_count=count,
members=manager.transaction_indices(set(item)))
                for (item, count) in itemsets.items()
            }
            for (length, itemsets) in large_itemsets.items()
        }
        return itemsets_out, len(manager)
    return large_itemsets, len(manager)

class Rule(object):
    # Number of decimals used for printing
    _decimals = 3

    def __init__(
        self,
        lhs: tuple,
        rhs: tuple,
        count_full: int = 0,
        count_lhs: int = 0,
        count_rhs: int = 0,
        num_transactions: int = 0,
    ):
        self.lhs = lhs  # antecedent
        self.rhs = rhs  # consequent
        self.count_full = count_full
        self.count_lhs = count_lhs
        self.count_rhs = count_rhs
        self.num_transactions = num_transactions

    @property
    def confidence(self):
        try:
            return self.count_full / self.count_lhs
        except ZeroDivisionError:
            return None
        except AttributeError:
            return None

    @property
    def support(self):
        try:
            return self.count_full / self.num_transactions
        except ZeroDivisionError:
            return None
        except AttributeError:
            return None

    @property
    def lift(self):
        try:
            observed_support = self.count_full / self.num_transactions
            prod_counts = self.count_lhs * self.count_rhs
            expected_support = prod_counts / self.num_transactions**2
```

```
            return observed_support / expected_support
        except ZeroDivisionError:
            return None
        except AttributeError:
            return None

    @property
    def conviction(self):
        try:
            eps = 10e-10  # Avoid zero division
            prob_not_rhs = 1 - self.count_rhs / self.num_transactions
            prob_not_rhs_given_lhs = 1 - self.confidence
            return prob_not_rhs / (prob_not_rhs_given_lhs + eps)
        except ZeroDivisionError:
            return None
        except AttributeError:
            return None

    @property
    def rpf(self):
        try:
            return self.confidence * self.support
        except ZeroDivisionError:
            return None
        except AttributeError:
            return None

    @staticmethod
    def _pf(s):
        return "{" + ", ".join(str(k) for k in s) + "}"
    def __repr__(self):
        return "{} -> {}".format(self._pf(self.lhs), self._pf(self.rhs))
    def __str__(self):
        conf = "conf: {0:.3f}".format(self.confidence)
        supp = "supp: {0:.3f}".format(self.support)
        lift = "lift: {0:.3f}".format(self.lift)
        conv = "conv: {0:.3f}".format(self.conviction)
        return "{} -> {} ({}, {}, {}, {})".format(self._pf(self.lhs), self._pf(self.rhs),
conf, supp, lift, conv)
    def __eq__(self, other):
        return (set(self.lhs) == set(other.lhs)) and (set(self.rhs) == set(other.rhs))
    def __hash__(self):
        return hash(frozenset(self.lhs + self.rhs))
    def __len__(self):
        return len(self.lhs + self.rhs)


def generate_rules_simple(itemsets: typing.Dict[int, typing.Dict],min_confidence:
float,num_transactions: int,):
    for size in itemsets.keys():
        if size < 2:
            continue
        yielded: set = set()
        yielded_add = yielded.add
        for itemset in itemsets[size].keys():

            # Generate rules
```

```
            for result in _genrules(itemset, itemset, itemsets, min_confidence,
num_transactions):
                if result in yielded:
                    continue
                else:
                    yielded_add(result)
                    yield result


def _genrules(l_k, a_m, itemsets, min_conf, num_transactions):
    def count(itemset):
        return itemsets[len(itemset)][itemset]
    for a_m in itertools.combinations(a_m, len(a_m) - 1):
        confidence = count(l_k) / count(a_m)
        if confidence < min_conf:
            continue
        rhs = set(l_k).difference(set(a_m))
        rhs = tuple(sorted(rhs))
        yield Rule(a_m, rhs, count(l_k), count(a_m), count(rhs), num_transactions)
        if len(a_m) <= 1:
            continue
        yield from _genrules(l_k, a_m, itemsets, min_conf, num_transactions)


def generate_rules_apriori(itemsets: typing.Dict[int, typing.Dict[tuple,
int]],min_confidence: float,num_transactions: int,verbosity: int = 0,):
    if not ((0 <= min_confidence <= 1) and isinstance(min_confidence, numbers.Number)):
        raise ValueError("`min_confidence` must be a number between 0 and 1.")

    if not ((num_transactions >= 0) and isinstance(num_transactions, numbers.Number)):
        raise ValueError("`num_transactions` must be a number greater than 0.")

    def count(itemset):
        return itemsets[len(itemset)][itemset]

    if verbosity > 0:
        print("Generating rules from itemsets.")
    for size in itemsets.keys():
        if size < 2:
            continue

        if verbosity > 0:
            print(" Generating rules of size {}.".format(size))

        for itemset in itemsets[size].keys():
            for removed in itertools.combinations(itemset, 1):
                remaining = set(itemset).difference(set(removed))
                lhs = tuple(sorted(remaining))
                conf = count(itemset) / count(lhs)
                if conf >= min_confidence:
                    yield Rule(
                        lhs,
                        removed,
                        count(itemset),
                        count(lhs),
                        count(removed),
                        num_transactions,
                    )
```

```
            H_1 = list(itertools.combinations(itemset, 1))
            yield from _ap_genrules(itemset, H_1, itemsets, min_confidence,
num_transactions)

    if verbosity > 0:
        print("Rule generation terminated.\n")

def _ap_genrules(itemset: tuple,H_m: typing.List[tuple],itemsets: typing.Dict[int,
typing.Dict[tuple, int]],min_conf: float,num_transactions: int,):
    def count(itemset):
        return itemsets[len(itemset)][itemset]
    if len(itemset) <= (len(H_m[0]) + 1):
        return

    H_m = list(apriori_gen(H_m))
    H_m_copy = H_m.copy()

    for h_m in H_m:
        lhs = tuple(sorted(set(itemset).difference(set(h_m))))
        if (count(itemset) / count(lhs)) >= min_conf:
            yield Rule(
                lhs,
                h_m,
                count(itemset),
                count(lhs),
                count(h_m),
                num_transactions,
            )
        else:
            H_m_copy.remove(h_m)
    if H_m_copy:
        yield from _ap_genrules(itemset, H_m_copy, itemsets, min_conf, num_transactions)

def apriori(
    transactions: typing.Iterable[typing.Union[set, tuple, list]],
    min_support: float = 0.5,
    min_confidence: float = 0.5,
    max_length: int = 8,
    verbosity: int = 0,
    output_transaction_ids: bool = False,
):
    itemsets, num_trans = itemsets_from_transactions(
        transactions,
        min_support,
        max_length,
        verbosity,
        output_transaction_ids=True,
    )

    itemsets_raw = {
        length: {item: counter.itemset_count for (item, counter) in itemsets.items()}
        for (length, itemsets) in itemsets.items()
    }
    rules = generate_rules_apriori(itemsets_raw, min_confidence, num_trans, verbosity)

    if output_transaction_ids:
        return itemsets, list(rules)
    else:
```

```
        return itemsets_raw, list(rules)
```

设置support（支持度）和confidence（置信度）的最小阈值为0.03和0.7，找出满足条件的频繁项集与关联规则：

- **由于数据集数量较多，故设置min_support ≥ 0.03**
- **为保证获得置信度较高的关联规则，设置min_confidence ≥ 0.7**

```
itemsets, rules = apriori(transactions, min_support=0.03,  min_confidence=0.7)
print('频繁项集：')
display(itemsets)
```

```
频繁项集：
```


```
{1: {('low_points',): 43760,
  ('cheap',): 38746,
  ('Willamette Valley_region',): 2271,
  ('@paulgwine\xa0_twitter',): 9453,
  ('expensive',): 35546,
  ('Pinot Noir_variety',): 8639,
  ('@wineschach_twitter',): 10355,
  ('@kerinokeefe_twitter',): 9849,
  ('@vossroger_twitter',): 14424,
  ('@vboone_twitter',): 9502,
  ('Cabernet Sauvignon_variety',): 5043,
  ('@mattkettmann_twitter',): 6236,
  ('Chardonnay_variety',): 6971,
  ('Malbec_variety',): 2335,
  ('Mendoza_region',): 2272,
  ('Red Blend_variety',): 5845,
  ('Columbia Valley (WA)_region',): 4108,
  ('@wawinereport_twitter',): 4752,
  ('Sauvignon Blanc_variety',): 2248,
  ('Bordeaux-style Red Blend_variety',): 4498,
  ('@gordone_cellars_twitter',): 4165,
  ('@JoeCz_twitter',): 3324,
  ('Rosé_variety',): 2467,
  ('Syrah_variety',): 2613,
  ('high_points',): 30532},
 2: {('@gordone_cellars_twitter', 'cheap'): 2755,
  ('@gordone_cellars_twitter', 'low_points'): 2500,
  ('@kerinokeefe_twitter', 'Red Blend_variety'): 2355,
  ('@kerinokeefe_twitter', 'cheap'): 4524,
  ('@kerinokeefe_twitter', 'expensive'): 5325,
  ('@kerinokeefe_twitter', 'high_points'): 3539,
  ('@kerinokeefe_twitter', 'low_points'): 6310,
  ('@mattkettmann_twitter', 'expensive'): 4123,
  ('@mattkettmann_twitter', 'high_points'): 3648,
  ('@mattkettmann_twitter', 'low_points'): 2588,
  ('@paulgwine\xa0_twitter', 'Pinot Noir_variety'): 2714,
  ('@paulgwine\xa0_twitter', 'cheap'): 4638,
  ('@paulgwine\xa0_twitter', 'expensive'): 4815,
  ('@paulgwine\xa0_twitter', 'high_points'): 4373,
  ('@paulgwine\xa0_twitter', 'low_points'): 5080,
  ('@vboone_twitter', 'cheap'): 2996,
```

```
 ('@vboone_twitter', 'expensive'): 6506,
 ('@vboone_twitter', 'high_points'): 5063,
 ('@vboone_twitter', 'low_points'): 4439,
 ('@vossroger_twitter', 'Bordeaux-style Red Blend_variety'): 3151,
 ('@vossroger_twitter', 'cheap'): 8342,
 ('@vossroger_twitter', 'expensive'): 6082,
 ('@vossroger_twitter', 'high_points'): 5637,
 ('@vossroger_twitter', 'low_points'): 8787,
 ('@wawinereport_twitter', 'expensive'): 2651,
 ('@wawinereport_twitter', 'low_points'): 2850,
 ('@wineschach_twitter', 'Mendoza_region'): 2272,
 ('@wineschach_twitter', 'cheap'): 7741,
 ('@wineschach_twitter', 'expensive'): 2614,
 ('@wineschach_twitter', 'high_points'): 2388,
 ('@wineschach_twitter', 'low_points'): 7967,
 ('Bordeaux-style Red Blend_variety', 'cheap'): 2292,
 ('Bordeaux-style Red Blend_variety', 'low_points'): 2665,
 ('Cabernet Sauvignon_variety', 'expensive'): 3195,
 ('Cabernet Sauvignon_variety', 'high_points'): 2315,
 ('Cabernet Sauvignon_variety', 'low_points'): 2728,
 ('Chardonnay_variety', 'cheap'): 3330,
 ('Chardonnay_variety', 'expensive'): 3641,
 ('Chardonnay_variety', 'high_points'): 3175,
 ('Chardonnay_variety', 'low_points'): 3796,
 ('Columbia Valley (WA)_region', 'cheap'): 2379,
 ('Columbia Valley (WA)_region', 'low_points'): 2481,
 ('Pinot Noir_variety', 'expensive'): 6790,
 ('Pinot Noir_variety', 'high_points'): 5079,
 ('Pinot Noir_variety', 'low_points'): 3560,
 ('Red Blend_variety', 'cheap'): 3083,
 ('Red Blend_variety', 'expensive'): 2762,
 ('Red Blend_variety', 'low_points'): 3767,
 ('Rosé_variety', 'cheap'): 2246,
 ('cheap', 'high_points'): 7599,
 ('cheap', 'low_points'): 31147,
 ('expensive', 'high_points'): 22933,
 ('expensive', 'low_points'): 12613},
3: {('@kerinokeefe_twitter', 'cheap', 'low_points'): 3843,
 ('@kerinokeefe_twitter', 'expensive', 'high_points'): 2858,
 ('@kerinokeefe_twitter', 'expensive', 'low_points'): 2467,
 ('@mattkettmann_twitter', 'expensive', 'high_points'): 3016,
 ('@paulgwine\xa0_twitter', 'cheap', 'low_points'): 3295,
 ('@paulgwine\xa0_twitter', 'expensive', 'high_points'): 3030,
 ('@vboone_twitter', 'cheap', 'low_points'): 2235,
 ('@vboone_twitter', 'expensive', 'high_points'): 4302,
 ('@vossroger_twitter', 'cheap', 'low_points'): 6783,
 ('@vossroger_twitter', 'expensive', 'high_points'): 4078,
 ('@wineschach_twitter', 'cheap', 'low_points'): 6992,
 ('Chardonnay_variety', 'cheap', 'low_points'): 2746,
 ('Chardonnay_variety', 'expensive', 'high_points'): 2591,
 ('Pinot Noir_variety', 'expensive', 'high_points'): 4769,
 ('Red Blend_variety', 'cheap', 'low_points'): 2563}}
```

3. 导出关联规则，计算其支持度和置信度，并使用Lift（提升度）和Conviction（确信度）来评价

- Conf(置信度)：包含X和Y的事务数与所有包含X的事务数之比，也就是当项集X出现时，项集Y同时出现的概率
- Sup(支持度)：事务集中同时包含X和Y的事务数与所有事务数之比
- Lift(提升度)：越表明X和Y的相关性，lift<1负相关，lift=1独立，lift>1正相关
- Conv(确信度)：表示X出现而Y不出现的概率，也就是规则预测错误的概率,它的值越大，表明X、Y的独立性越小

```
import csv
pd.set_option('display.float_format',lambda x : '%.6f' % x)
pd.set_option('max_colwidth',100)
print('导出的关联规则与评价指标：')
rules_rhs = filter(lambda rule: (len(rule.lhs) >= 1), rules)
with open('result.csv', 'wt',encoding="utf-8") as f:
    f_csv = csv.writer(f, delimiter=',')
    f_csv.writerow(['rule','conf','sup','lift','conv'])
    for rule in sorted(rules_rhs, key=lambda rule: rule.lift,reverse = True):
        f_csv.writerow([str(rule).split('(')
[0],rule.confidence,rule.support,rule.lift,rule.conviction])

pd.read_csv('result.csv')
```

```
导出的关联规则与评价指标：
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	rule	conf	sup	lift	conv
0	{Mendoza_region} -> {@wineschach_twitter}	1.000000	0.030582	7.174505	860617563.129274
1	{Bordeaux-style Red Blend_variety} -> {@vossroger_twitter}	0.700534	0.042414	3.608156	2.690943
2	{Pinot Noir_variety, high_points} -> {expensive}	0.938964	0.064193	1.962458	8.544789
3	{@mattkettmann_twitter, expensive} -> {high_points}	0.731506	0.040597	1.779938	2.193820
4	{@vboone_twitter, high_points} -> {expensive}	0.849694	0.057907	1.775881	3.469829
5	{Rosé_variety} -> {cheap}	0.910418	0.030232	1.745644	5.341037
6	{Chardonnay_variety, expensive} -> {high_points}	0.711618	0.034876	1.731544	2.042522
7	{@mattkettmann_twitter, high_points} -> {expensive}	0.826754	0.040597	1.727937	3.010389
8	{Pinot Noir_variety, expensive} -> {high_points}	0.702356	0.064193	1.709009	1.978968
9	{Chardonnay_variety, high_points} -> {expensive}	0.816063	0.034876	1.705591	2.835409
10	{@kerinokeefe_twitter, high_points} -> {expensive}	0.807573	0.038470	1.687847	2.710306
11	{@wineschach_twitter, low_points} -> {cheap}	0.877620	0.094115	1.682758	3.909659
12	{Pinot Noir_variety} -> {expensive}	0.785971	0.091396	1.642698	2.436752
13	{high_points} -> {expensive}	0.751114	0.308687	1.569846	2.095481
14	{@wineschach_twitter, cheap} -> {low_points}	0.903242	0.094115	1.533448	4.247452
15	{@vossroger_twitter, high_points} -> {expensive}	0.723434	0.054892	1.511996	1.885761
16	{@vossroger_twitter, low_points} -> {cheap}	0.771936	0.091302	1.480118	2.097933
17	{@kerinokeefe_twitter, cheap} -> {low_points}	0.849469	0.051728	1.442157	2.730164
18	{@wineschach_twitter} -> {cheap}	0.747562	0.104197	1.433383	1.895367

	rule	conf	sup	lift	conv
19	{Red Blend_variety, cheap} -> {low_points}	0.831333	0.034499	1.411367	2.436595
20	{Chardonnay_variety, cheap} -> {low_points}	0.824625	0.036962	1.399977	2.343390
21	{Chardonnay_variety, low_points} -> {cheap}	0.723393	0.036962	1.387042	1.729759
22	{@vossroger_twitter, cheap} -> {low_points}	0.813114	0.091302	1.380436	2.199061
23	{low_points} -> {cheap}	0.711769	0.419251	1.364753	1.659998
24	{cheap} -> {low_points}	0.803877	0.419251	1.364753	2.095481
25	{@wineschach_twitter} -> {low_points}	0.769387	0.107239	1.306199	1.782087
26	{@vboone_twitter, cheap} -> {low_points}	0.745995	0.030084	1.266486	1.617970
27	{@paulgwine _twitter, cheap} -> {low_points}	0.710436	0.044352	1.206117	1.419280

4. 对挖掘出的27条关联规则进行分析

首先对于X、Y项集中项数都为1的情况:

1. 葡萄产地为Mendoza的红酒 => 品鉴师@wineschach的关联置信度为1，葡萄品种为Bordeaux-style Red Blend_variety => 品鉴师@vossroger置信度0.7，即**该产地和葡萄品种的红酒大部分由这两位品鉴师品尝**
2. 品鉴师@wineschach => 价格cheap、打分low_points的置信度均大于0.7，lift1.3值左右，说明**他所品尝的红酒都较便宜，且打分较低**
3. 葡萄品种Rosé_variety => 价格cheap置信度0.9，说明**以该品种葡萄为原料的红酒价格普遍较低**
4. 葡萄品种Pinot Noir_variety => 价格expensive的置信度0.78，说明**以该品种葡萄为原料的红酒价格较高**
5. 打分high_points => 价格expensive的置信度0.75，**高分酒价格较高，符合常理**
6. 打分low_points => 价格cheap置信度0.71，价格cheap => 打分low_points置信度0.8,**低分酒便宜，便宜酒低分，也符合常理**

```
rules_rhs = filter(lambda rule: len(rule.lhs) == 1 and len(rule.rhs) == 1, rules)
with open('result_1-1.csv', 'wt',encoding="utf-8") as f:
    f_csv = csv.writer(f, delimiter=',')
    f_csv.writerow(['rule','conf','sup','lift','conv'])
    for rule in sorted(rules_rhs, key=lambda rule: rule.lift,reverse = True):
        f_csv.writerow([str(rule).split('(')
[0],rule.confidence,rule.support,rule.lift,rule.conviction])

pd.read_csv('result_1-1.csv')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	rule	conf	sup	lift	conv
0	{Mendoza_region} -> {@wineschach_twitter}	1.000000	0.030582	7.174505	860617563.129274
1	{Bordeaux-style Red Blend_variety} -> {@vossroger_twitter}	0.700534	0.042414	3.608156	2.690943
2	{Rosé_variety} -> {cheap}	0.910418	0.030232	1.745644	5.341037
3	{Pinot Noir_variety} -> {expensive}	0.785971	0.091396	1.642698	2.436752
4	{high_points} -> {expensive}	0.751114	0.308687	1.569846	2.095481
5	{@wineschach_twitter} -> {cheap}	0.747562	0.104197	1.433383	1.895367
6	{low_points} -> {cheap}	0.711769	0.419251	1.364753	1.659998
7	{cheap} -> {low_points}	0.803877	0.419251	1.364753	2.095481
8	{@wineschach_twitter} -> {low_points}	0.769387	0.107239	1.306199	1.782087

对于X、Y项集中项数大于1的情况:

1. {葡萄种类，高价(低价)} => {高分(低分)} 之类
2. {品鉴师，高价(低价)} => {高分(低分)} 、{品鉴师，高分(低分)} => {高价(低价)} 之类

```
rules_rhs = filter(lambda rule: len(rule.lhs) > 1, rules)
with open('result_2-1.csv', 'wt',encoding="utf-8") as f:
    f_csv = csv.writer(f, delimiter=',')
    f_csv.writerow(['rule','conf','sup','lift','conv'])
    for rule in sorted(rules_rhs, key=lambda rule: rule.lift,reverse = True):
        f_csv.writerow([str(rule).split('(')
[0],rule.confidence,rule.support,rule.lift,rule.conviction])

pd.read_csv('result_2-1.csv')
```

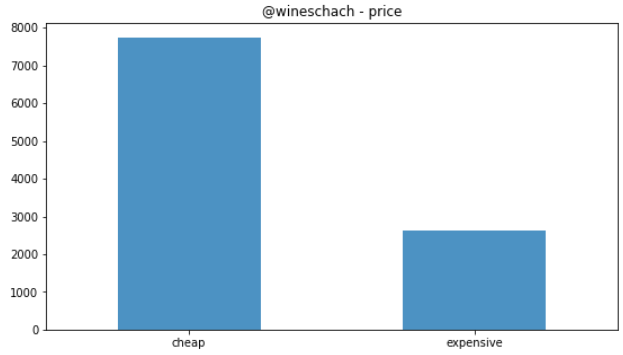
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

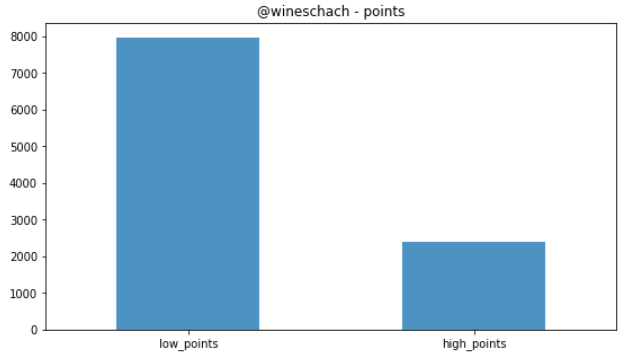
	rule	conf	sup	lift	conv
0	{Pinot Noir_variety, high_points} -> {expensive}	0.938964	0.064193	1.962458	8.544789
1	{@mattkettmann_twitter, expensive} -> {high_points}	0.731506	0.040597	1.779938	2.193820
2	{@vboone_twitter, high_points} -> {expensive}	0.849694	0.057907	1.775881	3.469829
3	{Chardonnay_variety, expensive} -> {high_points}	0.711618	0.034876	1.731544	2.042522
4	{@mattkettmann_twitter, high_points} -> {expensive}	0.826754	0.040597	1.727937	3.010389
5	{Pinot Noir_variety, expensive} -> {high_points}	0.702356	0.064193	1.709009	1.978968
6	{Chardonnay_variety, high_points} -> {expensive}	0.816063	0.034876	1.705591	2.835409
7	{@kerinokeefe_twitter, high_points} -> {expensive}	0.807573	0.038470	1.687847	2.710306
8	{@wineschach_twitter, low_points} -> {cheap}	0.877620	0.094115	1.682758	3.909659
9	{@wineschach_twitter, cheap} -> {low_points}	0.903242	0.094115	1.533448	4.247452
10	{@vossroger_twitter, high_points} -> {expensive}	0.723434	0.054892	1.511996	1.885761
11	{@vossroger_twitter, low_points} -> {cheap}	0.771936	0.091302	1.480118	2.097933
12	{@kerinokeefe_twitter, cheap} -> {low_points}	0.849469	0.051728	1.442157	2.730164
13	{Red Blend_variety, cheap} -> {low_points}	0.831333	0.034499	1.411367	2.436595
14	{Chardonnay_variety, cheap} -> {low_points}	0.824625	0.036962	1.399977	2.343390
15	{Chardonnay_variety, low_points} -> {cheap}	0.723393	0.036962	1.387042	1.729759
16	{@vossroger_twitter, cheap} -> {low_points}	0.813114	0.091302	1.380436	2.199061
17	{@vboone_twitter, cheap} -> {low_points}	0.745995	0.030084	1.266486	1.617970
18	{@paulgwine _twitter, cheap} -> {low_points}	0.710436	0.044352	1.206117	1.419280

四.可视化展示与验证

1 以品鉴师@wineschach => 价格cheap、打分low_points为例，可视化查看具体情况进行检验，符合挖掘结果

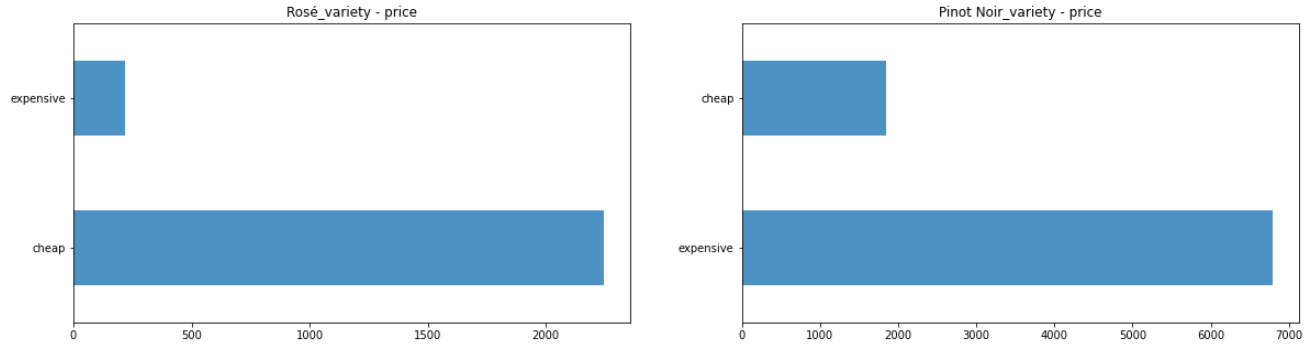
```
fig = plt.figure(figsize=(20,5))
fig.add_subplot(1,2,1)
A[A['taster_twitter_handle']=='@wineschach_twitter']
['price'].value_counts().plot(kind='bar',alpha=0.8)
plt.xticks(rotation=0)
plt.title('@wineschach - price')
fig.add_subplot(1,2,2)
A[A['taster_twitter_handle']=='@wineschach_twitter']
['points'].value_counts().plot(kind='bar',alpha=0.8)
plt.xticks(rotation=0)
plt.title('@wineschach - points')
plt.show()
```





2 可视化检验两种葡萄品种(variety)与价格(cheap,expensive)之间的关系，符合挖掘结果

```
fig = plt.figure(figsize=(20,5))
fig.add_subplot(1,2,1)
A.groupby('variety').get_group('Rosé_variety').price.value_counts().plot(kind='barh',alph
a=0.8)
plt.xticks(rotation=0)
plt.title('Rosé_variety - price')
fig.add_subplot(1,2,2)
A.groupby('variety').get_group('Pinot
Noir_variety').price.value_counts().plot(kind='barh',alpha=0.8)
plt.xticks(rotation=0)
plt.title('Pinot Noir_variety - price')
plt.show()
```



3 可视化检验价格(cheap,expensive)与打分(low_points,high_points)之间的关系，符合挖掘结果

```
A.groupby('price').points.value_counts().plot(kind='line',figsize=
(15,6),linestyle='-.',linewidth=3,alpha=0.8)
```

```
<AxesSubplot:xlabel='price,points'>
```