21200889_assignment_1_code

May 28, 2022

1 Code Submission

Jesus Cortina Bernal - 21200889

Code Adapted from Ruane, E. (2022) Example Jupyter Notebook for Assignment 1

1.1 Section 1

[1]:
```
# import required packages here
import math
import pandas as pd
import collections
import ast
import itertools as i_tools
import mlxtend.preprocessing as mlxt_pre
import mlxtend.frequent_patterns as mlxt_fp
import statistics
```

[2]:
```
# load the basket.csv file into a pandas DataFrame (df)
bskt_raw = pd.read_csv('21200889_assignment_1_data/21200889_assignment_1_basket.
 ↪csv')

# Use df.shape to see the number of rows and columns in the df
print(bskt_raw.shape)
```

```
(15206, 12)
```

[3]:
```
# Use df.head() to see the first 8 rows of the df
bskt_raw.head(8)
```

[3]:

	ID	0	1	2	3	4	5	6	7	8
0	1	whole milk	eggs	salty snack	NaN	NaN	NaN	NaN	NaN	NaN
1	2	whole milk	eggs	white bread	yogurt	NaN	NaN	NaN	NaN	NaN
2	3	whole milk	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	4	whole milk	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	5	whole milk	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	6	whole milk	eggs	rolls/buns	NaN	NaN	NaN	NaN	NaN	NaN
6	7	whole milk	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
7   8  whole milk  eggs  whipped/sour cream     NaN  NaN  NaN  NaN  NaN  NaN

     9   10
0  NaN  NaN
1  NaN  NaN
2  NaN  NaN
3  NaN  NaN
4  NaN  NaN
5  NaN  NaN
6  NaN  NaN
7  NaN  NaN
```

[4]:
```
# Set the index of the df to the Transaction_ID column
bskt_raw.set_index('ID', inplace=True)

# Use df.head() to see the first 3 rows of the df
bskt_raw.head(3)
```

[4]:
```
             0     1            2       3    4    5    6    7    8    9   10
ID
1   whole milk  eggs  salty snack     NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
2   whole milk  eggs  white bread  yogurt  NaN  NaN  NaN  NaN  NaN  NaN  NaN
3   whole milk  eggs          NaN     NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
```

[5]:
```
# Use the pandas describe method to get an overview of the dataset
bskt_raw.describe()
```

[5]:
```
                0           1           2           3           4  \
count       15206       15204        5218        2398         886
unique        141         143         134         128         111
top     whole milk  whole milk  whole milk  whole milk  whole milk
freq         1418        1128         308         147          49

                5       6           7             8    9           10
count         477     281         198            51    1           1
unique         96      80          65            35    1           1
top     rolls/buns  yogurt  whole milk  shopping bags  jam  newspapers
freq           25      19          17             4    1           1
```

[6]:
```
#Replacing NAN values with strings and grabbing the columns with items
bskt_raw.fillna("NaN", inplace = True)
base_cols = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
```

[7]:
```
# create a list of lists, where each list contains the items in a given␣
 ↪transaction
bskt_raw["Shop_List"] = bskt_raw[base_cols].values.tolist()
bskt_lol = []
```

```
for i in range(1, len(bskt_raw) + 1):
    bskt_lol.append(bskt_raw.loc[i, 'Shop_List'])

# view the first transaction in the list
print(bskt_lol[0])
print(type(bskt_lol[0]))
print(len(bskt_lol))
```

```
['whole milk', 'eggs', 'salty snack', 'NaN', 'NaN', 'NaN', 'NaN', 'NaN', 'NaN',
'NaN', 'NaN']
<class 'list'>
15206
```

[8]:
```
# from your list of lists, create a flattened list that contains all items␣
 ↪purchased
bskt_flat = [item for transaction in bskt_lol for item in transaction]

# view the legnth of the list
print(len(bskt_flat))
```

```
167266
```

[9]:
```
# create a list of unique items from the flattened list
goods = list(set(bskt_flat))

# remove any elements that items of interest (like empty strings or nan values)
while "NaN" in bskt_flat:
    bskt_flat.remove("NaN")

goods.remove("NaN")

# print out how many unique items we have
print("# of items ", len(goods))
```

```
# of items  148
```

[10]:
```
# generate the itemset permutations up to 2-itemsets
rules = list(i_tools.permutations(goods, 2))

# print out the number of rules
print('# of rules:',len(rules))

# print some of the elements of the list of rules
print(*rules[:6])
```

```
# of rules: 21756
('onions', 'rubbing alcohol') ('onions', 'ham') ('onions', 'beef') ('onions',
'canned fish') ('onions', 'bags') ('onions', 'cereals')
```

[11]:
```
# generate the frequency counts for each unique item in the dataset
item_freq = collections.Counter(bskt_flat)

# show the most common
item_freq.most_common(1)
```

[11]:
```
[('whole milk', 3102)]
```

[12]:
```
# one-hot encode the data and show the first 5 rows of the resulting df
# don't forget to drop the 'nan' value
encoder = mlxt_pre.TransactionEncoder().fit(bskt_lol)
onehot = encoder.transform(bskt_lol)
bskt_onehot = pd.DataFrame(onehot, columns = encoder.columns_).drop('NaN',␣
 ↪axis=1)
bskt_onehot.head()
```

[12]:

	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	\
0	False	False	False	False	False	
1	False	False	False	False	False	
2	False	False	False	False	False	
3	False	False	False	False	False	
4	False	False	False	False	False	

	bathroom cleaner	beef	berries	beverages	bottled beer	…	\
0	False	False	False	False	False	…	
1	False	False	False	False	False	…	
2	False	False	False	False	False	…	
3	False	False	False	False	False	…	
4	False	False	False	False	False	…	

	tropical fruit	turkey	vinegar	waffles	whipped/sour cream	whisky	\
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	

	white bread	white wine	whole milk	yogurt
0	False	False	True	False
1	True	False	True	True
2	False	False	True	False
3	False	False	True	False
4	False	False	True	False

[5 rows x 148 columns]

[13]:
```
# Generate frequent itemsets with a minimum support of 0.5%
bskt_itemsets = mlxt_fp.apriori(bskt_onehot, min_support=0.005,␣
 ↪use_colnames=True)

# view the itemsets sorted by most frequent
bskt_itemsets.sort_values(by=['support'], ascending=False)
```

[13]:

	support	itemsets
77	0.191109	(whole milk)
45	0.128568	(other vegetables)
2	0.116336	(berries)
59	0.108181	(rolls/buns)
66	0.095226	(soda)
..	…	…
130	0.005195	(whipped/sour cream, whole milk)
50	0.005130	(photo/film)
38	0.005130	(mayonnaise)
79	0.005130	(beef, berries)
105	0.005064	(jam, whole milk)

[133 rows x 2 columns]

How many itemsets are generated for the following support thresholds?: * 0.1% * 0.5% * 1% * 10%

[14]:
```
#For 0.1%
q1 = mlxt_fp.apriori(bskt_onehot, min_support=0.001, use_colnames=True)

#For 0.5%
q2 = mlxt_fp.apriori(bskt_onehot, min_support=0.005, use_colnames=True)

#For 1%
q3 = mlxt_fp.apriori(bskt_onehot, min_support=0.01, use_colnames=True)

#For 10%
q4 = mlxt_fp.apriori(bskt_onehot, min_support=0.1, use_colnames=True)

#Results
print("# of items per support threshold:" + "\n \
    At 0.1% : ", len(q1), "\n \
    At 0.5% : ", len(q2), "\n \
    At 1% : " , len(q3) , "\n \
    At 10% : ", len(q4))
```

```
# of items per support threshold:
     At 0.1% :  762
     At 0.5% :  133
     At 1% :  68
     At 10% :  4
```

[15]:
```
# generate association rules with a confidence threshold of 10%
bskt_rules = mlxt_fp.association_rules(bskt_itemsets, metric='confidence',␣
 ↪min_threshold=0.1)

# how many rules are generated?
print("# of rules ", len(bskt_rules))
```

```
# of rules  42
```

[16]:
```
# print out the rules sorted by lift
print(bskt_rules.sort_values(by=['lift'], ascending=False)
      .drop(columns=['antecedent support', 'consequent support', 'conviction']))
```

```
             antecedents         consequents   support  confidence      lift  \
21                (eggs)        (whole milk)  0.015520    0.312169  1.633464
26     (root vegetables)  (other vegetables)  0.013087    0.171997  1.337790
25    (other vegetables)   (root vegetables)  0.013087    0.101790  1.337790
28         (white bread)  (other vegetables)  0.006116    0.171587  1.334602
9                (yogurt)           (berries)  0.012429    0.140000  1.203414
10              (berries)            (yogurt)  0.012429    0.106840  1.203414
2          (bottled beer)           (berries)  0.005590    0.123367  1.060442
20                (eggs)  (other vegetables)  0.006708    0.134921  1.049413
0                 (beef)  (other vegetables)  0.009930    0.133274  1.036610
6               (pastry)           (berries)  0.005787    0.113256  0.973529
11         (bottled beer)        (whole milk)  0.008089    0.178520  0.934126
40          (white bread)        (whole milk)  0.006313    0.177122  0.926811
23           (newspapers)        (whole milk)  0.006708    0.174061  0.910798
5               (berries)  (other vegetables)  0.013218    0.113624  0.883764
4     (other vegetables)           (berries)  0.013218    0.102813  0.883764
3        (bottled water)           (berries)  0.006116    0.101751  0.874629
7                 (soda)           (berries)  0.009667    0.101519  0.872642
14              (butter)        (whole milk)  0.005590    0.161290  0.843971
36       (shopping bags)        (whole milk)  0.007431    0.158931  0.831626
8               (berries)        (whole milk)  0.018480    0.158847  0.831185
34          (rolls/buns)        (whole milk)  0.016770    0.155015  0.811136
17           (chocolate)        (whole milk)  0.007300    0.153953  0.805577
15                (cake)        (whole milk)  0.005393    0.152700  0.799022
32           (pip fruit)        (whole milk)  0.007300    0.152055  0.795645
41              (yogurt)        (whole milk)  0.013482    0.151852  0.794583
33                (pork)        (whole milk)  0.006116    0.150729  0.788710
27                (soda)  (other vegetables)  0.009601    0.100829  0.784246
29    (other vegetables)        (whole milk)  0.019203    0.149361  0.781548
30          (whole milk)  (other vegetables)  0.019203    0.100482  0.781548
24          (rolls/buns)  (other vegetables)  0.010851    0.100304  0.780165
1                 (beef)        (whole milk)  0.011048    0.148279  0.775887
18        (citrus fruit)        (whole milk)  0.007694    0.147170  0.770084
19              (coffee)        (whole milk)  0.006774    0.146933  0.768845
```

```
31              (pastry)        (whole milk)  0.007431    0.145431  0.760986
13         (brown bread)        (whole milk)  0.006116    0.145086  0.759179
22                 (jam)        (whole milk)  0.005064    0.144195  0.754517
12       (bottled water)        (whole milk)  0.008615    0.143326  0.749971
38      (tropical fruit)        (whole milk)  0.009667    0.142996  0.748245
35     (root vegetables)        (whole milk)  0.010719    0.140882  0.737180
16         (canned beer)        (whole milk)  0.006445    0.139601  0.730480
37                (soda)        (whole milk)  0.013153    0.138122  0.722738
39  (whipped/sour cream)        (whole milk)  0.005195    0.122291  0.639903

    leverage
21  0.006019
26  0.003304
25  0.003304
28  0.001533
9   0.002101
10  0.002101
2   0.000319
20  0.000316
0   0.000351
6  -0.000157
11 -0.000570
40 -0.000499
23 -0.000657
5  -0.001739
4  -0.001739
3  -0.000877
7  -0.001411
14 -0.001033
36 -0.001505
8  -0.003753
34 -0.003905
17 -0.001762
15 -0.001356
32 -0.001875
41 -0.003485
33 -0.001638
27 -0.002641
29 -0.005367
30 -0.005367
24 -0.003058
1  -0.003191
18 -0.002297
19 -0.002037
31 -0.002334
13 -0.001940
22 -0.001648
12 -0.002872
```

```
38 -0.003253
35 -0.003822
16 -0.002378
37 -0.005046
39 -0.002924
```

1.2 Code for Generation of Rules

[17]:
```
#Mean and Median for item counts

item_dict = {}

for t in bskt_lol:
    for i in goods:
        if i in t and i not in item_dict:
            item_dict[i] = 1
        elif i in t and i in item_dict:
            item_dict[i] += 1

mean = statistics.mean(item_dict.values())
median = statistics.median(item_dict.values())
print("Mean :", mean, " and Median :", median)
```

```
Mean : 263.6216216216216  and Median : 87.5
```

[18]:
```
#Obtaining the list of the most frequent items
most_freq = sorted(item_dict.items(), key = lambda k:k[1], reverse = True)
print(*most_freq[:11])
```

```
('whole milk', 2906) ('other vegetables', 1955) ('berries', 1769) ('rolls/buns',
1645) ('soda', 1448) ('yogurt', 1350) ('root vegetables', 1157) ('beef', 1133)
('tropical fruit', 1028) ('bottled water', 914) ('citrus fruit', 795)
```

[19]:
```
#Cell for investigating the support of individual items
item_inp = input("Please write the item that you want to check the count/
 ↪support of: ")
print("For item", item_inp, " the Count is",
      item_dict[item_inp], " and the Support is",
      round(item_dict[item_inp]/15206,6))
```

```
Please write the item that you want to check the count/support of: whole milk
For item whole milk  the Count is 2906  and the Support is 0.191109
```

[20]:
```
#Working with Support 1%, Confidence 10%
bskt_r1 = mlxt_fp.apriori(bskt_onehot, min_support=0.01, use_colnames=True)
rules_1 = mlxt_fp.association_rules(bskt_r1, metric='confidence',␣
 ↪min_threshold=0.10)
```

```
rules_1["Cosine IS"] = (rules_1.lift * rules_1.support) ** (1/2)

print(rules_1.sort_values(by=['lift', 'confidence', 'support'], ascending=False)
      .drop(columns=['conviction']))
```

```
           antecedents         consequents  antecedent support  \
6               (eggs)        (whole milk)            0.049717
9    (root vegetables)  (other vegetables)            0.076088
8   (other vegetables)   (root vegetables)            0.128568
4              (yogurt)           (berries)            0.088781
5             (berries)            (yogurt)            0.116336
2             (berries)  (other vegetables)            0.116336
1   (other vegetables)           (berries)            0.128568
3             (berries)        (whole milk)            0.116336
12        (rolls/buns)        (whole milk)            0.108181
15             (yogurt)        (whole milk)            0.088781
10  (other vegetables)        (whole milk)            0.128568
11        (whole milk)  (other vegetables)            0.191109
7         (rolls/buns)  (other vegetables)            0.108181
0               (beef)        (whole milk)            0.074510
13   (root vegetables)        (whole milk)            0.076088
14              (soda)        (whole milk)            0.095226

    consequent support   support  confidence      lift  leverage  Cosine IS
6             0.191109  0.015520    0.312169  1.633464  0.006019   0.159222
9             0.128568  0.013087    0.171997  1.337790  0.003304   0.132316
8             0.076088  0.013087    0.101790  1.337790  0.003304   0.132316
4             0.116336  0.012429    0.140000  1.203414  0.002101   0.122301
5             0.088781  0.012429    0.106840  1.203414  0.002101   0.122301
2             0.128568  0.013218    0.113624  0.883764 -0.001739   0.108083
1             0.116336  0.013218    0.102813  0.883764 -0.001739   0.108083
3             0.191109  0.018480    0.158847  0.831185 -0.003753   0.123935
12            0.191109  0.016770    0.155015  0.811136 -0.003905   0.116630
15            0.191109  0.013482    0.151852  0.794583 -0.003485   0.103500
10            0.191109  0.019203    0.149361  0.781548 -0.005367   0.122507
11            0.128568  0.019203    0.100482  0.781548 -0.005367   0.122507
7             0.128568  0.010851    0.100304  0.780165 -0.003058   0.092008
0             0.191109  0.011048    0.148279  0.775887 -0.003191   0.092586
13            0.191109  0.010719    0.140882  0.737180 -0.003822   0.088894
14            0.191109  0.013153    0.138122  0.722738 -0.005046   0.097498
```

[21]:
```
#Working with Support 0.5%, Confidence 10%
bskt_r2 = mlxt_fp.apriori(bskt_onehot, min_support=0.005, use_colnames=True)
rules_2 = mlxt_fp.association_rules(bskt_r2, metric='confidence',␣
 ↪min_threshold=0.10)

rules_2["Cosine IS"] = (rules_2.lift * rules_2.support) ** (1/2)
```

```
rules_2f = rules_2["support"] <= 0.01

print(rules_2[rules_2f].sort_values(by=['Cosine IS', 'confidence', 'support'],␣
 ↪ascending=False)
      .drop(columns=['conviction']))
```

```
            antecedents         consequents  antecedent support  \
0                (beef)  (other vegetables)            0.074510
7                (soda)           (berries)            0.095226
28         (white bread)  (other vegetables)            0.035644
11        (bottled beer)        (whole milk)            0.045311
27               (soda)  (other vegetables)            0.095226
38      (tropical fruit)        (whole milk)            0.067605
20               (eggs)  (other vegetables)            0.049717
12       (bottled water)        (whole milk)            0.060108
36       (shopping bags)        (whole milk)            0.046758
23          (newspapers)        (whole milk)            0.038537
2         (bottled beer)           (berries)            0.045311
18        (citrus fruit)        (whole milk)            0.052282
17           (chocolate)        (whole milk)            0.047415
40         (white bread)        (whole milk)            0.035644
32           (pip fruit)        (whole milk)            0.048007
31              (pastry)        (whole milk)            0.051098
6               (pastry)           (berries)            0.051098
3        (bottled water)           (berries)            0.060108
19              (coffee)        (whole milk)            0.046100
33                (pork)        (whole milk)            0.040576
14              (butter)        (whole milk)            0.034657
16         (canned beer)        (whole milk)            0.046166
13         (brown bread)        (whole milk)            0.042154
15                (cake)        (whole milk)            0.035315
22                 (jam)        (whole milk)            0.035118
39  (whipped/sour cream)        (whole milk)            0.042483

    consequent support   support  confidence      lift  leverage  Cosine IS
0             0.128568  0.009930    0.133274  1.036610  0.000351   0.101459
7             0.116336  0.009667    0.101519  0.872642 -0.001411   0.091848
28            0.128568  0.006116    0.171587  1.334602  0.001533   0.090346
11            0.191109  0.008089    0.178520  0.934126 -0.000570   0.086926
27            0.128568  0.009601    0.100829  0.784246 -0.002641   0.086775
38            0.191109  0.009667    0.142996  0.748245 -0.003253   0.085050
20            0.128568  0.006708    0.134921  1.049413  0.000316   0.083901
12            0.191109  0.008615    0.143326  0.749971 -0.002872   0.080380
36            0.191109  0.007431    0.158931  0.831626 -0.001505   0.078613
23            0.191109  0.006708    0.174061  0.910798 -0.000657   0.078163
2             0.116336  0.005590    0.123367  1.060442  0.000319   0.076992
```

```
18            0.191109  0.007694    0.147170  0.770084 -0.002297   0.076976
17            0.191109  0.007300    0.153953  0.805577 -0.001762   0.076684
40            0.191109  0.006313    0.177122  0.926811 -0.000499   0.076493
32            0.191109  0.007300    0.152055  0.795645 -0.001875   0.076210
31            0.191109  0.007431    0.145431  0.760986 -0.002334   0.075200
6             0.116336  0.005787    0.113256  0.973529 -0.000157   0.075060
3             0.116336  0.006116    0.101751  0.874629 -0.000877   0.073138
19            0.191109  0.006774    0.146933  0.768845 -0.002037   0.072166
33            0.191109  0.006116    0.150729  0.788710 -0.001638   0.069453
14            0.191109  0.005590    0.161290  0.843971 -0.001033   0.068686
16            0.191109  0.006445    0.139601  0.730480 -0.002378   0.068614
13            0.191109  0.006116    0.145086  0.759179 -0.001940   0.068141
15            0.191109  0.005393    0.152700  0.799022 -0.001356   0.065642
22            0.191109  0.005064    0.144195  0.754517 -0.001648   0.061812
39            0.191109  0.005195    0.122291  0.639903 -0.002924   0.057658
```

[22]:
```
#Working with Support 0.1%, Confidence 10%, Sorted by Cosine
bskt_r3 = mlxt_fp.apriori(bskt_onehot, min_support=0.001, use_colnames=True)
rules_3 = mlxt_fp.association_rules(bskt_r3, metric='confidence',␣
 ↪min_threshold=0.1)

rules_3["Cosine IS"] = (rules_3.lift * rules_3.support) ** (1/2)

rules_3f = rules_3["support"] <= 0.005

print(rules_3[rules_3f].sort_values(by=['Cosine IS'], ascending=False).head(20)
.drop(columns=['conviction']))
```

```
                                        antecedents  \
92                                        (ketchup)
93                                        (mustard)
53                                       (pet care)
52                                       (cat food)
215                       (eggs, other vegetables)
217                 (root vegetables, whole milk)
216                       (eggs, root vegetables)
1                                          (flour)
2                                  (baking powder)
195                (other vegetables, white bread)
196            (other vegetables, root vegetables)
183                       (eggs, other vegetables)
197                 (white bread, root vegetables)
184                       (eggs, root vegetables)
151                       (beef, other vegetables)
214  (other vegetables, root vegetables, whole milk)
185            (other vegetables, root vegetables)
152                       (beef, root vegetables)
212            (eggs, other vegetables, whole milk)
```

```
153              (other vegetables, root vegetables)

                       consequents  antecedent support  consequent support  \
92                       (mustard)            0.003749            0.007168
93                       (ketchup)            0.007168            0.003749
53                      (cat food)            0.005656            0.013679
52                      (pet care)            0.013679            0.005656
215   (root vegetables, whole milk)            0.006708            0.010719
217        (eggs, other vegetables)            0.010719            0.006708
216  (other vegetables, whole milk)            0.004669            0.019203
1                   (baking powder)            0.011574            0.009141
2                           (flour)            0.009141            0.011574
195               (root vegetables)            0.006116            0.076088
196                   (white bread)            0.013087            0.035644
183               (root vegetables)            0.006708            0.076088
197              (other vegetables)            0.004406            0.128568
184              (other vegetables)            0.004669            0.128568
151               (root vegetables)            0.009930            0.076088
214                           (eggs)            0.002565            0.049717
185                           (eggs)            0.013087            0.049717
152              (other vegetables)            0.006839            0.128568
212               (root vegetables)            0.001907            0.076088
153                           (beef)            0.013087            0.074510

      support  confidence       lift  leverage  Cosine IS
92   0.001381    0.368421  51.396427  0.001354   0.266421
93   0.001381    0.192661  51.396427  0.001354   0.266421
53   0.001644    0.290698  21.251677  0.001567   0.186921
52   0.001644    0.120192  21.251677  0.001567   0.186921
215  0.001250    0.186275  17.377240  0.001178   0.147353
217  0.001250    0.116564  17.377240  0.001178   0.147353
216  0.001250    0.267606  13.935655  0.001160   0.131957
1    0.001315    0.113636  12.431328  0.001209   0.127869
2    0.001315    0.143885  12.431328  0.001209   0.127869
195  0.002696    0.440860   5.794054  0.002231   0.124990
196  0.002696    0.206030   5.780248  0.002230   0.124841
183  0.002762    0.411765   5.411663  0.002252   0.122259
197  0.002696    0.611940   4.759675  0.002130   0.113285
184  0.002762    0.591549   4.601073  0.002162   0.112732
151  0.003091    0.311258   4.090746  0.002335   0.112446
214  0.001250    0.487179   9.799010  0.001122   0.110652
185  0.002762    0.211055   4.245114  0.002111   0.108283
152  0.003091    0.451923   3.515060  0.002212   0.104234
212  0.001250    0.655172   8.610676  0.001104   0.103726
153  0.003091    0.236181   3.169785  0.002116   0.098982
```

[23]:
```
#Support 0.1%, Confidence 10%, Sorted by Lift
print(rules_3[rules_3f].sort_values(by=['lift'], ascending=False).head(20)
.drop(columns=['conviction']))
```

```
                                        antecedents  \
92                                        (ketchup)
93                                        (mustard)
53                                       (pet care)
52                                       (cat food)
215                       (eggs, other vegetables)
217                  (root vegetables, whole milk)
216                        (eggs, root vegetables)
1                                           (flour)
2                                   (baking powder)
214  (other vegetables, root vegetables, whole milk)
212            (eggs, other vegetables, whole milk)
213             (eggs, root vegetables, whole milk)
195                 (other vegetables, white bread)
196             (other vegetables, root vegetables)
181                      (other vegetables, chicken)
183                         (eggs, other vegetables)
182                       (root vegetables, chicken)
197                  (white bread, root vegetables)
159                              (beef, white bread)
184                        (eggs, root vegetables)

                       consequents  antecedent support  consequent support  \
92                        (mustard)            0.003749            0.007168
93                        (ketchup)            0.007168            0.003749
53                       (cat food)            0.005656            0.013679
52                       (pet care)            0.013679            0.005656
215   (root vegetables, whole milk)            0.006708            0.010719
217       (eggs, other vegetables)            0.010719            0.006708
216  (other vegetables, whole milk)            0.004669            0.019203
1                   (baking powder)            0.011574            0.009141
2                           (flour)            0.009141            0.011574
214                          (eggs)            0.002565            0.049717
212              (root vegetables)            0.001907            0.076088
213             (other vegetables)            0.001513            0.128568
195              (root vegetables)            0.006116            0.076088
196                  (white bread)            0.013087            0.035644
181              (root vegetables)            0.003749            0.076088
183              (root vegetables)            0.006708            0.076088
182             (other vegetables)            0.002433            0.128568
197             (other vegetables)            0.004406            0.128568
159              (root vegetables)            0.003157            0.076088
184             (other vegetables)            0.004669            0.128568
```

```
      support  confidence       lift  leverage  Cosine IS
92   0.001381    0.368421  51.396427  0.001354   0.266421
93   0.001381    0.192661  51.396427  0.001354   0.266421
53   0.001644    0.290698  21.251677  0.001567   0.186921
52   0.001644    0.120192  21.251677  0.001567   0.186921
215  0.001250    0.186275  17.377240  0.001178   0.147353
217  0.001250    0.116564  17.377240  0.001178   0.147353
216  0.001250    0.267606  13.935655  0.001160   0.131957
1    0.001315    0.113636  12.431328  0.001209   0.127869
2    0.001315    0.143885  12.431328  0.001209   0.127869
214  0.001250    0.487179   9.799010  0.001122   0.110652
212  0.001250    0.655172   8.610676  0.001104   0.103726
213  0.001250    0.826087   6.425309  0.001055   0.089602
195  0.002696    0.440860   5.794054  0.002231   0.124990
196  0.002696    0.206030   5.780248  0.002230   0.124841
181  0.001578    0.421053   5.533731  0.001293   0.093456
183  0.002762    0.411765   5.411663  0.002252   0.122259
182  0.001578    0.648649   5.045193  0.001265   0.089235
197  0.002696    0.611940   4.759675  0.002130   0.113285
159  0.001118    0.354167   4.654674  0.000878   0.072138
184  0.002762    0.591549   4.601073  0.002162   0.112732
```

[24]:
```
#Working with Support 0.05%, Confidence 10%, Sorted by Cosine
bskt_r4 = mlxt_fp.apriori(bskt_onehot, min_support=0.0005, use_colnames=True)
rules_4 = mlxt_fp.association_rules(bskt_r4, metric='confidence',␣
 ↪min_threshold=0.1)

rules_4["Cosine IS"] = (rules_4.lift * rules_4.support) ** (1/2)

rules_4f = rules_4["support"] <= 0.001

print(rules_4[rules_4f].sort_values(by=['Cosine IS'], ascending=False).head(20)
.drop(columns=['conviction']))
```

```
                                antecedents  \
236                           (baking powder)
234                     (flour, whole milk)
235               (baking powder, whole milk)
228                           (flour, eggs)
230                 (eggs, baking powder)
602                   (beef, white bread)
601                (beef, root vegetables)
603          (other vegetables, white bread)
604           (root vegetables, white bread)
615                        (yogurt, chicken)
627              (root vegetables, chicken)
614                        (chicken, berries)
```

```
626               (other vegetables, chicken)
624                   (eggs, root vegetables)
625                           (eggs, chicken)
613                 (chicken, tropical fruit)
598     (beef, other vegetables, white bread)
633                   (eggs, root vegetables)
634             (white bread, root vegetables)
597  (beef, other vegetables, root vegetables)

                           consequents  antecedent support  \
236                 (flour, whole milk)            0.009141
234                     (baking powder)            0.002565
235                             (flour)            0.002170
228                     (baking powder)            0.001118
230                             (flour)            0.000921
602  (other vegetables, root vegetables)            0.003157
601     (other vegetables, white bread)            0.006839
603              (beef, root vegetables)            0.006116
604             (beef, other vegetables)            0.004406
615           (berries, tropical fruit)            0.002367
627             (eggs, other vegetables)            0.002433
614             (yogurt, tropical fruit)            0.002565
626              (eggs, root vegetables)            0.003749
624         (other vegetables, chicken)            0.004669
625  (other vegetables, root vegetables)            0.001710
613                   (yogurt, berries)            0.001841
598                   (root vegetables)            0.001315
633     (other vegetables, white bread)            0.004669
634             (eggs, other vegetables)            0.004406
597                        (white bread)            0.003091

     consequent support   support  confidence       lift  leverage  Cosine IS
236            0.002565  0.000986    0.107914  42.075263  0.000963   0.203728
234            0.009141  0.000986    0.384615  42.075263  0.000963   0.203728
235            0.011574  0.000986    0.454545  39.271694  0.000961   0.196824
228            0.009141  0.000592    0.529412  57.915362  0.000582   0.185144
230            0.011574  0.000592    0.642857  55.541396  0.000581   0.181310
602            0.013087  0.000986    0.312500  23.878769  0.000945   0.153477
601            0.006116  0.000986    0.144231  23.582506  0.000945   0.152522
603            0.006839  0.000986    0.161290  23.582506  0.000945   0.152522
604            0.009930  0.000986    0.223881  22.545221  0.000943   0.149130
615            0.006642  0.000526    0.222222  33.456546  0.000510   0.132672
627            0.006708  0.000526    0.216216  32.233174  0.000510   0.130223
614            0.006445  0.000526    0.205128  31.828362  0.000510   0.129403
626            0.004669  0.000526    0.140351  30.058809  0.000509   0.125754
624            0.003749  0.000526    0.112676  30.058809  0.000509   0.125754
625            0.013087  0.000526    0.307692  23.511403  0.000504   0.111218
613            0.012429  0.000526    0.285714  22.987150  0.000503   0.109971
```

```
598            0.076088  0.000986    0.750000   9.856958  0.000886   0.098607
633            0.006116  0.000526    0.112676  18.423141  0.000498   0.098451
634            0.006708  0.000526    0.119403  17.800410  0.000497   0.096773
597            0.035644  0.000986    0.319149   8.953835  0.000876   0.093982
```

[25]:
```
#h-confidence checker
h_ante = input("Antecedent item: ")
h_cons = input("Consequent item: ")


print(rules_3.loc[(rules_3['antecedents'] == {h_ante}) &␣
↪(rules_3['consequents'] == {h_cons})]
    .drop(columns = ['antecedent support', 'consequent support', 'conviction',␣
↪'Cosine IS',
                     'lift', 'leverage', 'support']))
```

```
Antecedent item: whole milk
Consequent item: eggs
Empty DataFrame
Columns: [antecedents, consequents, confidence]
Index: []
```

[26]:
```
#Checker for h-confidence with only antecedent
h_ante = input("Antecedent item: ")

print(rules_3.loc[(rules_3['antecedents'] == {h_ante})].drop(columns =␣
↪['conviction']))
```

```
Antecedent item: whole milk
      antecedents         consequents  antecedent support  consequent support  \
119  (whole milk)  (other vegetables)            0.191109            0.128568

      support  confidence      lift  leverage  Cosine IS
119  0.019203    0.100482  0.781548 -0.005367   0.122507
```

1.3 Section 2 - OPTION

This section details one of thee options. For the other two options, see the assignment instructions document (*Instructions_Assignment_1.pdf*).

Your task is to implement the Apriori algorithm from "scratch" using the same dataset as Section 1.

You can read the data into a dataframe and use pandas methods but you cannot use mlxtend or another similar package. You should write functions to implement the steps of the algorithm to create frequent itemsets.

Calculate the support, confidence, and lift for single itemset rules.

1.4 OPTION 3 APPLICATIONS

1.5 POKEMON

[27]:
```
#Importing the Pokemon Data Set
poke_raw = pd.read_csv('21200889_assignment_1_data/21200889_assignment_1_poke.
 ↪csv')
poke_raw
```

[27]:

	ID	1	2	3	4	5	\
0	1	Amoonguss	Incineroar	Landorus	Moltres	Regieleki	
1	2	Grimmsnarl-G	Landorus	Moltres	Regieleki	Registeel	
2	3	Blastoise-G	Incineorar	Landorus	Registeel	Rillaboom-G	
3	4	Grimmsnarl	Landorus	Porygon2	Registeel	Torkoal	
4	5	Dusclops	Glastrier	Incineroar	Regieleki	Tapu Fini	
5	6	Celesteela	Ferrothorn	Garchomp	Grimmsnarl-G	Incineroar	
6	7	Coalossal-G	Dragapult	Incineroar	Moltres	Rillaboom-G	
7	8	Incineroar	Landorus	Regieleki	Togekiss	Urshifu-G	
8	9	Grimmsnarl-G	Landorus	Porygon2	Registeel	Torkoal	
9	10	Amoonguss	Glastrier	Incineroar	Landorus	Moltres	
10	11	Blastoise-G	Clefairy	Landorus	Regieleki	Registeel	
11	12	Incineroar	Metagross	Moltres	Raikou	Tapu Fini	
12	13	Incineroar	Indeedee	Porygon2	Stakataka	Torkoal	
13	14	Dusclops	Glastrier	Hatterene-G	Indeedee	Torkoal	
14	15	Blastoise-G	Coalossal-G	Dragapult	Hatterene-G	Indeedee	

	6
0	Tapu Fini
1	Tapu Fini
2	Thundurus
3	Venusaur-G
4	Urshifu-G
5	Tapu Fini
6	Urshifu-G
7	Weezing
8	Venusaur-G
9	Regieleki
10	Spectrier
11	Whimsicott
12	Venusaur-G
13	Venusaur-G
14	Urshifu-G

[28]:
```
#Cleaning the Data Set
poke_cols = [str(i + 1) for i in range(6)]
poke_raw = poke_raw.set_index('ID')

# create a list of lists
```

```
poke_raw["Poke_List"] = poke_raw[poke_cols].values.tolist()
poke_lol = []
for i in range(1, len(poke_raw) + 1):
    poke_lol.append(poke_raw.loc[i, 'Poke_List'])
```

[29]:
```
#Creating the encoder

encoder_p = mlxt_pre.TransactionEncoder().fit(poke_lol)
onehot_p = encoder_p.transform(poke_lol)
poke_onehot = pd.DataFrame(onehot_p, columns = encoder_p.columns_)
poke_onehot
```

[29]:

	Amoonguss	Blastoise-G	Celesteela	Clefairy	Coalossal-G	Dragapult	\
0	True	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	True	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
5	False	False	True	False	False	False	
6	False	False	False	False	True	True	
7	False	False	False	False	False	False	
8	False	False	False	False	False	False	
9	True	False	False	False	False	False	
10	False	True	False	True	False	False	
11	False	False	False	False	False	False	
12	False	False	False	False	False	False	
13	False	False	False	False	False	False	
14	False	True	False	False	True	True	

	Dusclops	Ferrothorn	Garchomp	Glastrier	…	Spectrier	Stakataka	\
0	False	False	False	False	…	False	False	
1	False	False	False	False	…	False	False	
2	False	False	False	False	…	False	False	
3	False	False	False	False	…	False	False	
4	True	False	False	True	…	False	False	
5	False	True	True	False	…	False	False	
6	False	False	False	False	…	False	False	
7	False	False	False	False	…	False	False	
8	False	False	False	False	…	False	False	
9	False	False	False	True	…	False	False	
10	False	False	False	False	…	True	False	
11	False	False	False	False	…	False	False	
12	False	False	False	False	…	False	True	
13	True	False	False	True	…	False	False	
14	False	False	False	False	…	False	False	

	Tapu Fini	Thundurus	Togekiss	Torkoal	Urshifu-G	Venusaur-G	Weezing	\

```
0        True      False     False    False      False       False    False
1        True      False     False    False      False       False    False
2       False       True     False    False      False       False    False
3       False      False     False     True      False        True    False
4        True      False     False    False       True       False    False
5        True      False     False    False      False       False    False
6       False      False     False    False       True       False    False
7       False      False      True    False       True       False     True
8       False      False     False     True      False        True    False
9       False      False     False    False      False       False    False
10      False      False     False    False      False       False    False
11       True      False     False    False      False       False    False
12      False      False     False     True      False        True    False
13      False      False     False     True      False        True    False
14      False      False     False    False       True       False    False

    Whimsicott
0        False
1        False
2        False
3        False
4        False
5        False
6        False
7        False
8        False
9        False
10       False
11        True
12       False
13       False
14       False

[15 rows x 34 columns]
```

[30]:
```
#Applying Apriori and Rules
poke_sets = mlxt_fp.apriori(poke_onehot, min_support=0.25, use_colnames=True)

poke_rules = mlxt_fp.association_rules(poke_sets, metric='confidence',␣
 ↪min_threshold=0.1)

# print out the rules sorted by lift
print(poke_rules.sort_values(by=['lift'], ascending=False)
      .drop(columns=['antecedent support', 'consequent support', 'conviction']))
```

```
     antecedents   consequents   support  confidence    lift  leverage
10  (Venusaur-G)     (Torkoal)  0.266667    1.000000  3.7500  0.195556
```

```
11     (Torkoal)  (Venusaur-G)  0.266667    1.000000  3.7500  0.195556
8    (Registeel)    (Landorus)  0.333333    1.000000  1.8750  0.155556
9     (Landorus)   (Registeel)  0.333333    0.625000  1.8750  0.155556
6     (Landorus)   (Regieleki)  0.333333    0.625000  1.5625  0.120000
7    (Regieleki)    (Landorus)  0.333333    0.833333  1.5625  0.120000
0      (Moltres)  (Incineroar)  0.266667    0.800000  1.5000  0.088889
1   (Incineroar)     (Moltres)  0.266667    0.500000  1.5000  0.088889
4    (Tapu Fini)  (Incineroar)  0.266667    0.800000  1.5000  0.088889
5   (Incineroar)   (Tapu Fini)  0.266667    0.500000  1.5000  0.088889
2   (Incineroar)   (Regieleki)  0.266667    0.500000  1.2500  0.053333
3    (Regieleki)  (Incineroar)  0.266667    0.666667  1.2500  0.053333
```

1.6 LEAGUE OF LEGENDS

[31]:
```
#Importing the League Data Set
league_raw = pd.read_csv('21200889_assignment_1_data/21200889_assignment_1_lol.
 ↪csv')
league_raw
```

[31]:
```
   ID                     1                          2  \
0   1      Boots of Swiftness      Shurelya's Battlesong
1   2  Staff of Flowing Water      Shurelya's Battlesong
2   3       Shard of True Ice      Shurelya's Battlesong
3   4       Shard of True Ice      Shurelya's Battlesong
4   5       Shard of True Ice  Locket of the Iron Solari
5   6   Shurelya's Battlesong    Bulwark of the Mountain

                          3                         4                 5
0   Bulwark of the Mountain                       NaN               NaN
1         Shard of True Ice  Ionian Boots of Lucidity               NaN
2  Ionian Boots of Lucidity                       NaN               NaN
3  Ionian Boots of Lucidity       Chemtech Putrifier               NaN
4  Ionian Boots of Lucidity   Staff of Flowing Water  Mikael's Crucible
5       Boots of Swiftness                Redemption      Ardent Censer
```

[32]:
```
#Cleaning the Data Set
league_cols = [str(i + 1) for i in range(5)]
league_raw = league_raw.set_index('ID')
league_raw = league_raw.fillna("NaN")

# create a list of lists
league_raw["LOL_List"] = league_raw[league_cols].values.tolist()
league_lol = []
for i in range(1, len(league_raw) + 1):
    league_lol.append(league_raw.loc[i, 'LOL_List'])
```

[33]:
```
#Creating the encoder

encoder_l = mlxt_pre.TransactionEncoder().fit(league_lol)
onehot_l = encoder_l.transform(league_lol)
league_onehot = pd.DataFrame(onehot_l, columns = encoder_l.columns_).
 ↪drop('NaN', axis=1)
league_onehot
```

[33]:

	Ardent Censer	Boots of Swiftness	Bulwark of the Mountain	\
0	False	True	True	
1	False	False	False	
2	False	False	False	
3	False	False	False	
4	False	False	False	
5	True	True	True	

	Chemtech Putrifier	Ionian Boots of Lucidity	Locket of the Iron Solari	\
0	False	False	False	
1	False	True	False	
2	False	True	False	
3	True	True	False	
4	False	True	True	
5	False	False	False	

	Mikael's Crucible	Redemption	Shard of True Ice	Shurelya's Battlesong	\
0	False	False	False	True	
1	False	False	True	True	
2	False	False	True	True	
3	False	False	True	True	
4	True	False	True	False	
5	False	True	False	True	

	Staff of Flowing Water
0	False
1	True
2	False
3	False
4	True
5	False

[34]:
```
#Applying Apriori and Rules
league_sets = mlxt_fp.apriori(league_onehot, min_support=0.4, use_colnames=True)

league_rules = mlxt_fp.association_rules(league_sets, metric='confidence',␣
 ↪min_threshold=0.1)

# print out the rules sorted by lift
```

```
print(league_rules.sort_values(by=['lift'], ascending=False)
      .drop(columns=['antecedent support', 'consequent support', 'conviction']))
```

```
                                        antecedents  \
0                                (Shard of True Ice)
1                           (Ionian Boots of Lucidity)
6           (Shurelya's Battlesong, Shard of True Ice)
7   (Shurelya's Battlesong, Ionian Boots of Lucidity)
10                               (Shard of True Ice)
11                          (Ionian Boots of Lucidity)
2                              (Shurelya's Battlesong)
4                              (Shurelya's Battlesong)
9                              (Shurelya's Battlesong)
3                           (Ionian Boots of Lucidity)
5                                (Shard of True Ice)
8         (Shard of True Ice, Ionian Boots of Lucidity)

                                        consequents   support  confidence  \
0                           (Ionian Boots of Lucidity)  0.666667        1.00
1                                (Shard of True Ice)  0.666667        1.00
6                           (Ionian Boots of Lucidity)  0.500000        1.00
7                                (Shard of True Ice)  0.500000        1.00
10  (Shurelya's Battlesong, Ionian Boots of Lucidity)  0.500000        0.75
11          (Shurelya's Battlesong, Shard of True Ice)  0.500000        0.75
2                           (Ionian Boots of Lucidity)  0.500000        0.60
4                                (Shard of True Ice)  0.500000        0.60
9         (Shard of True Ice, Ionian Boots of Lucidity)  0.500000        0.60
3                              (Shurelya's Battlesong)  0.500000        0.75
5                              (Shurelya's Battlesong)  0.500000        0.75
8                              (Shurelya's Battlesong)  0.500000        0.75

    lift  leverage
0    1.5  0.222222
1    1.5  0.222222
6    1.5  0.166667
7    1.5  0.166667
10   1.5  0.166667
11   1.5  0.166667
2    0.9 -0.055556
4    0.9 -0.055556
9    0.9 -0.055556
3    0.9 -0.055556
5    0.9 -0.055556
8    0.9 -0.055556
```

[ ]: