Association rules

February 17, 2020

0.1 Introduction

The goal of this assignment is to determine which SKUs appear to have relationships with each other, such that we can move them appropriately around the Dillard's store to increase revenues. The steps involved in this experiment our outlined below, with the appropriate code.

0.2 Understanding the data

To understand the data, I imported all of the data into Python using the pandas feature. Most of the data matched with the data schema, except the trnsact dataset which appeared to have an extra column. After looking at the data more, it was realized that the column was the sale price. Additionally, each of the datasets had a column with zeroes, which had to be removed.

[186]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from random import randint
from random import seed
from sklearn.model_selection import train_test_split
from tqdm import tqdm
```

[3]:
```
deptinfo = pd.read_csv("/Users/ArshyaSrinivas/Google Drive/Engineering/Junior␣
↪year/WINTER 2019/IEMS 308/association_rules/DillardsPOS/deptinfo.csv",␣
↪header = None)
```

[4]:
```
skst = pd.read_csv("/Users/ArshyaSrinivas/Google Drive/Engineering/Junior year/
↪WINTER 2019/IEMS 308/association_rules/DillardsPOS/skstinfo.csv", header =␣
↪None)
```

[190]:
```
%%capture
skuinfo = pd.read_csv("/Users/ArshyaSrinivas/Google Drive/Engineering/Junior␣
↪year/WINTER 2019/IEMS 308/association_rules/DillardsPOS/skuinfo.csv", header␣
↪= None, error_bad_lines=False);
```

[6]:
```
strinfo = pd.read_csv("/Users/ArshyaSrinivas/Google Drive/Engineering/Junior␣
↪year/WINTER 2019/IEMS 308/association_rules/DillardsPOS/strinfo.csv", header␣
↪= None)
```

[7]:
```
trnsact = pd.read_csv("/Users/ArshyaSrinivas/Google Drive/Engineering/Junior␣
↪year/WINTER 2019/IEMS 308/association_rules/DillardsPOS/trnsact.csv", header␣
↪= None)
```

[8]:
```
trnsact = trnsact.iloc[:,:-1]
```

[9]:
```
trnsact.columns = [ 'SKU','STORE', 'REGISTER',  'TRANNUM' , 'SEQ' ,␣
↪'SALEDATE','STYPE', 'QUANTITY','ORGPRICE', 'SPRICE', 'AMT', 'INTERID', 'MIC']
```

[10]:
```
#Renaming the rest of the columns
deptinfo = deptinfo.iloc[:,:-1]
skst = skst.iloc[:,:-1]
skuinfo = skuinfo.iloc[:,:-1]
strinfo = strinfo.iloc[:,:-1]

#Renaming more columns
deptinfo.columns = ['DEPT','DEPTDESC']
skuinfo.columns =␣
↪['SKU','DEPT','CLASSID','UPC','STYLE','COLOR','SIZE','PACKSIZE','VENDOR','BRAND']
strinfo.columns = ['STORE','CITY', 'STATE', 'ZIP']
skst.columns = ['SKU','STORE','COST','RETAIL']
```

Next, I looked at some of the values and the counts of different categories in the data. The first thing I looked at was the breakdown per state.

[188]:
```
#Determining the how many states there are
strinfo.STATE.value_counts()
```

[188]:
```
TX    79
FL    48
AR    27
AZ    26
OH    25
NC    24
LA    22
MO    20
TN    19
GA    16
OK    15
KS    15
CO    14
KY    14
AL    13
VA    10
SC     8
CA     8
MS     7
NM     6
UT     6
IA     5
```

```
NV     5
NE     4
ID     3
IL     3
NJ     3
MT     3
IN     2
NY     2
WY     1
Name: STATE, dtype: int64
```

Looking at the breakdown of the number of stores in each state, we can see how diverse the spread of stores are. After conducting some intial research, it was found that the state of California has the largest number of retail stores in the United States. It would be beneficial to apply the association rules found in this experiment to the stores in California, as these stores would appear to have the largest amount of competition. Hence, this data will be subsetted to the state of California.

[12]:
```
strinfo_CA = strinfo.loc[strinfo['STATE'] == "CA","STORE"]
str_CA_stores = [600,2600,3000,3600,6009,6109,6209,9106]
test = trnsact[trnsact.STORE.isin(str_CA_stores)]
```

[13]:
```
test.head(10)
```

[13]:

	SKU	STORE	REGISTER	TRANNUM	SEQ	SALEDATE	STYPE	QUANTITY	\
433	164	6109	580	2700	0	2005-08-25	P	1	
434	164	6109	580	2700	0	2005-08-25	P	1	
436	164	6209	550	4200	0	2005-08-27	P	1	
860	326	6009	160	2800	0	2004-08-19	P	1	
2430	450	6009	270	700	0	2005-08-23	P	1	
2431	450	6009	470	7200	0	2005-07-23	P	1	
2432	450	6009	580	3600	0	2005-07-18	P	1	
2433	450	6009	580	7400	0	2005-07-14	P	1	
2441	450	6109	20	2500	549308154	2005-06-18	P	1	
2442	450	6109	260	6600	0	2005-07-08	P	1	

	ORGPRICE	SPRICE	AMT	INTERID	MIC
433	20.0	20.00	20.00	255400008	822
434	20.0	20.00	20.00	255500008	822
436	20.0	20.00	20.00	266300013	822
860	72.0	72.00	72.00	682500004	106
2430	6.0	3.00	3.00	842300006	844
2431	6.0	3.00	3.00	427500010	844
2432	6.0	3.99	3.99	838500005	844
2433	6.0	3.99	3.99	930200006	844
2441	6.0	5.00	5.00	76400012	844
2442	6.0	3.99	3.99	231800007	844

Next, I decided to use the store with the highest number of data points for my analysis. I decided to choose the store with the highest value as it would provide the greatest competitive

advantage. Here, I am making an assumption that the larger the store is, the more customers are likely to come. By applying these association rules to the largest store, we can take advantage of a larger customer base.

[14]:
```
test.STORE.value_counts()
```

[14]:
```
6009    546321
6109    336128
6209     94297
Name: STORE, dtype: int64
```

[15]:
```
trnsact_store = test.loc[test['STORE'] == 6009]
```

The final subsetting done after exploring the data was determining the split of transactions that are either purchases or return transactions

[17]:
```
trnsact_store.STYPE.value_counts()
```

[17]:
```
P    502593
R     43728
Name: STYPE, dtype: int64
```

Here, it can be seen that the majority of the transactions are purchases. The data will be subsetted to only to this subset in order to only focus on what customers want to buy together initially. This will result in a fairer comparison of transactions.

[18]:
```
trnsact_purchase = trnsact_store.loc[trnsact_store['STYPE'] == "P"]
trnsact_purchase.head(10)
```

[18]:

	SKU	STORE	REGISTER	TRANNUM	SEQ	SALEDATE	STYPE	QUANTITY	\
860	326	6009	160	2800	0	2004-08-19	P	1	
2430	450	6009	270	700	0	2005-08-23	P	1	
2431	450	6009	470	7200	0	2005-07-23	P	1	
2432	450	6009	580	3600	0	2005-07-18	P	1	
2433	450	6009	580	7400	0	2005-07-14	P	1	
2957	497	6009	50	8000	0	2005-02-25	P	1	
2958	497	6009	60	20300	0	2005-02-26	P	1	
2959	497	6009	150	300	44809161	2005-02-01	P	1	
2960	497	6009	160	700	0	2005-05-06	P	1	
2962	497	6009	160	1900	922905343	2004-10-27	P	1	

	ORGPRICE	SPRICE	AMT	INTERID	MIC
860	72.0	72.00	72.00	682500004	106
2430	6.0	3.00	3.00	842300006	844
2431	6.0	3.00	3.00	427500010	844
2432	6.0	3.99	3.99	838500005	844
2433	6.0	3.99	3.99	930200006	844
2957	24.0	24.00	24.00	653300012	881
2958	24.0	24.00	24.00	367100016	881
2959	24.0	24.00	24.00	240300005	881
2960	24.0	24.00	24.00	404000008	881
2962	24.0	24.00	24.00	40000004	881

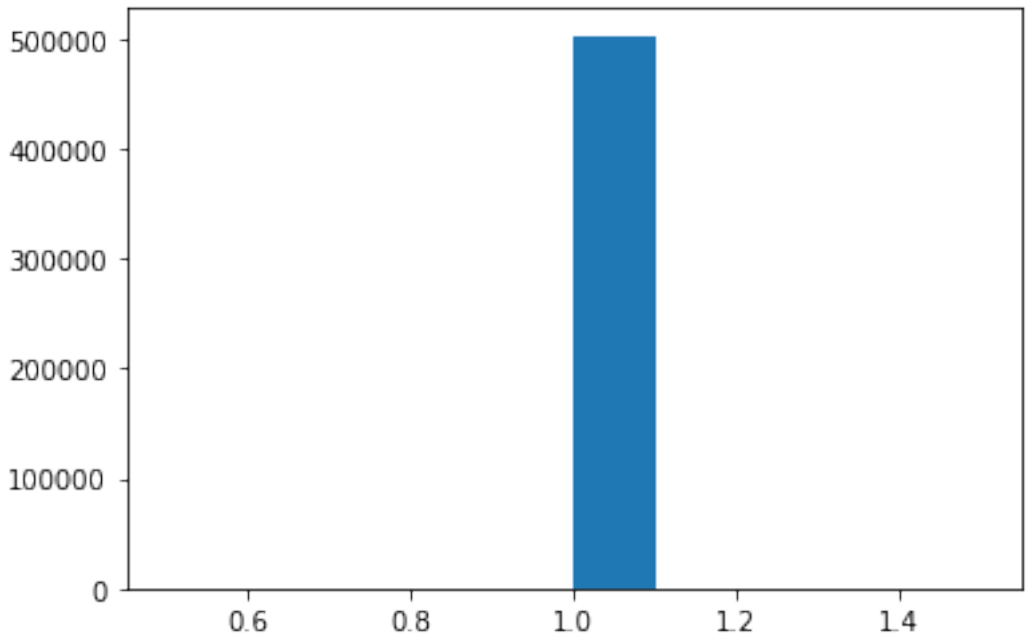
0.3 Exploratory data analysis

Since the data has been subsetted, some exploratory data analysis will be done on the transact dataset in order to look for underlying patterns.

The quantity section does not appear to have any split for this particular store and hence, we will disregard this. The first type of analysis that will be done is looking at the range of different quantities of a certain item purchased.

[136]:
```
plt.hist(trnsact_purchase.QUANTITY)
```

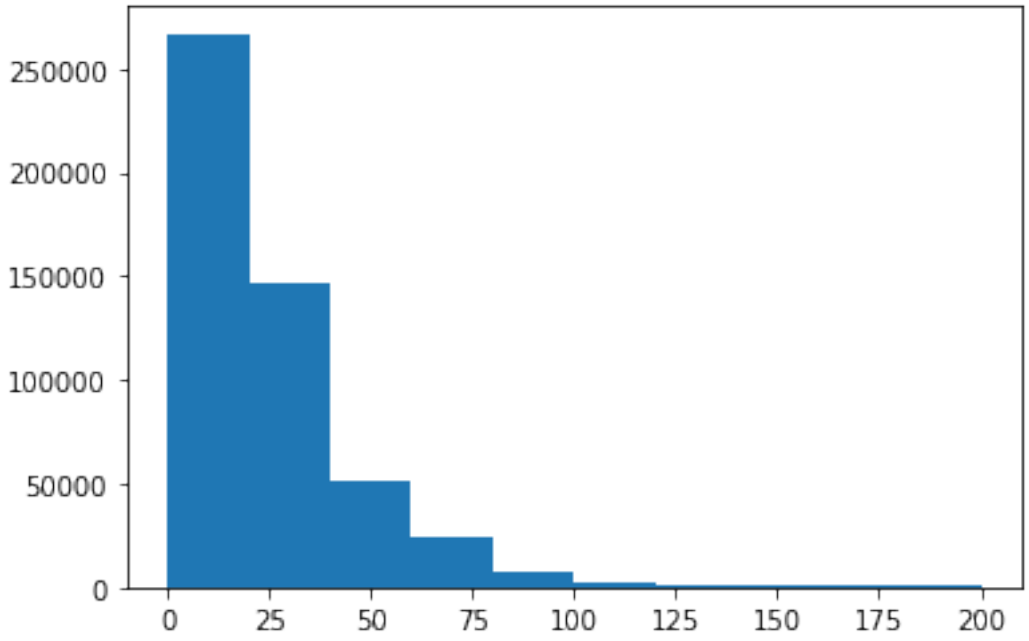
[136]:
```
(array([     0.,      0.,      0.,      0.,      0., 502593.,      0.,
             0.,      0.,      0.]),
 array([0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2, 1.3, 1.4, 1.5]),
 <a list of 10 Patch objects>)
```



Next, I looked at the spread of data for the sale price. From here, it appeared that the majority of the data is skewed to the right, with the majority of the prices less than 100 dollars. This gives a sense of how often more expensive items are purchased and can help us consider how to set the minimum support of the association rules later on in the experiment. Since there are definitely products that are considered to be "rare" but do generate a profit because they are relatively expensive, the minimum support of the experiment should be set low.

[144]:
```
plt.hist(trnsact_purchase.SPRICE, range=[0, 200])
```

[144]:
```
(array([266755., 146122.,  51268.,  24243.,   7775.,   2042.,   1369.,
            749.,    395.,    595.]),
 array([  0.,  20.,  40.,  60.,  80., 100., 120., 140., 160., 180., 200.]),
 <a list of 10 Patch objects>)
```



Looking deeper at prices, I identified whether the original price and the sale price of items are the same. If they are not, the compnay may have offered a discounted price, or were willing to take a cut in the price because they were desperate to sell the item. From here, it can be seen that approximately half of the subsetted data has a sale price that was less than the original sale price. It is interesting to note that this makes up half of the purchases and going forward, it should be kept in mind that a discounted price could have affected an association.

[142]:
```
sum(trnsact_purchase.ORGPRICE == trnsact_purchase.SPRICE)
```

[142]: 240997

I also took a look at whether there are any null values; it appears that there are no null values or missing values in the dataset that I subsetted.

[140]:
```
trnsact_purchase.isnull().values.any()
```

[140]: False

0.4 Completing association rules.

Now that the exploratory data analysis is complete, we can move forward with looking at the association rules within the data set.

[20]:
```
new_df = trnsact_purchase.loc[:,["REGISTER", "TRANNUM", "SKU"]]
```

One hot encoding is required to be done to determine which SKUs were purchased when. However, the computation power of my computer does not allow one hot encoding to be done for every single SKU. Hence, a subset of SKUs will be selected. This subset will randomly select 1000 SKUs and the analysis will be done on these 1000 SKUs. Since the assignment only requested 100 SKUs to be selected for moving around in the store, randomly selecting 1000 SKUs in this part

of the analysis gives the ability to have a large enough dataset to work with, but is constrained enough to receive results in a timely manner.

[40]:
```
%%capture

seed(1)

random_SKU = []

for i in tqdm(range(2000)):
    r = randint(0,unique_SKU.shape[0])
    if unique_SKU.get_value(r,0) not in random_SKU:
        random_SKU.append(unique_SKU.get_value(r,0))
```

```
  0%|          | 0/2000 [00:00<?,
?it/s]/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:8: FutureWarning: get_value is deprecated and
will be removed in a future release. Please use .at[] or .iat[] accessors
instead

/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:9: FutureWarning: get_value is deprecated and
will be removed in a future release. Please use .at[] or .iat[] accessors
instead
  if __name__ == '__main__':
100%|| 2000/2000 [00:00<00:00, 16766.45it/s]
```

[41]:
```
final_df = new_df[new_df['SKU'].isin(random_SKU)]
final_df
```

[41]:

	REGISTER	TRANNUM	SKU
119964	190	2300	9791
177036	290	3900	16036
177037	290	6000	16036
329386	240	200	26316
470341	290	1300	34005
470343	290	8000	34005
470344	290	8000	34005
470346	290	13900	34005
470347	290	14300	34005
670657	480	3600	47535
743041	380	2200	54148
743042	380	5500	54148
743043	380	10000	54148
756766	370	300	56585
781434	50	1800	57773
781435	80	2100	57773
781436	80	3300	57773
781437	240	1900	57773

838665	150	700	60697
838666	250	800	60697
858827	50	7500	64045
892225	230	1900	67364
1000869	780	200	76646
1074694	580	4000	82782
1088421	290	11500	84470
1123356	310	11600	87612
1157899	160	2800	90497
1157900	160	4100	90497
1157901	160	4900	90497
1157902	240	2300	90497
...	...	...	...
120168847	710	5000	9936627
120171166	360	801	9936751
120171167	360	900	9936751
120171169	360	3400	9936751
120171170	360	11400	9936751
120171171	370	2700	9936751
120171172	370	6900	9936751
120171173	530	8000	9936751
120171174	530	13600	9936751
120296820	460	800	9947889
120303547	270	2100	9948312
120384052	330	3600	9954563
120384053	340	600	9954563
120384054	340	1000	9954563
120391954	470	1700	9956428
120391955	470	1900	9956428
120391956	470	10100	9956428
120404734	580	4200	9956673
120407602	130	5500	9956869
120407603	140	3300	9956869
120550330	160	2800	9969033
120550332	240	3000	9969033
120550333	250	4300	9969033
120550334	780	1100	9969033
120605168	50	300	9976460
120694496	80	2800	9982466
120718191	50	500	9986460
120718192	110	4300	9986460
120718193	300	5400	9986460
120813028	360	11500	9992316

[6419 rows x 3 columns]

[42]:
```
#One-hot-encoding
onehot = pd.get_dummies(final_df['SKU'],prefix='sku')
df = pd.concat([final_df,onehot],axis=1)
df.drop(['SKU'],axis=1, inplace=True)
```

We would have to adjust our dataset in order to determine which of the items were ordered together. My assumption is that if items are ordered together, they would have the same TRANNUM number and would have the same REGISTER number. The data is now grouped together like this.

[43]:
```
finalData = df.groupby(['REGISTER','TRANNUM']).sum()
```

Before moving forward, we are going to split the data set into a training data set and a testing data set, so we can test the association rules on a subset of data after arriving to these rules. For this method, we will be training our model on 80% of our data and testing our model on 20% of our data. This ratio was chosen as we need a relatively large training set as we are determining patterns between a large number of SKUs; we need to reduce the probability of a SKU association happening by pure chance.

[145]:
```
training_data = finalData.iloc[:round(finalData.shape[0]*0.8),:]
testing_data = finalData.iloc[round(finalData.shape[0]*0.8)+1:,:]
```

When looking at the training dataset, it was noticed that a couple of products did not have binary values in their columns because of the group by function. Since we are only interested in seeing whether some products can be associated with each other (instead of looking at whether multiple of the same product are purchased at the same time), I decided to replace values that were more than one by one in the dataset.

[120]:
```
%%capture

training_data[training_data >= 1] = 1
```

```
/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Now, association rules will be applied to the training dataset.

[121]:
```
frequentItemsets_train = apriori(training_data, min_support=0.00001,␣
↪use_colnames=True)
```

[122]:
```
train_rules = association_rules(frequentItemsets_train, metric="lift",␣
↪min_threshold=1)
```

[123]:
```
train_rules.describe()
```

[123]:

	antecedent support	consequent support	support	confidence	\
count	297990.000000	297990.000000	297990.000000	297990.000000	
mean	0.000768	0.000768	0.000424	0.875454	

```
std              0.001388            0.001388       0.000022       0.273903
min              0.000423            0.000423       0.000423       0.012821
25%              0.000423            0.000423       0.000423       1.000000
50%              0.000423            0.000423       0.000423       1.000000
75%              0.000423            0.000423       0.000423       1.000000
max              0.033023            0.033023       0.002964       1.000000

                lift       leverage    conviction
count  297990.000000  297990.000000  2.979900e+05
mean     1803.234527       0.000423           inf
std       820.002148       0.000021           NaN
min         2.523504       0.000256  1.007841e+00
25%      1181.000000       0.000423           inf
50%      2362.000000       0.000423           inf
75%      2362.000000       0.000423           inf
max      2362.000000       0.002880           inf
```

[124]:
```
train_rules[train_rules['lift'] >= 100 ].describe()
```

[124]:
```
       antecedent support  consequent support        support     confidence  \
count       293284.000000       293284.000000  293284.000000  293284.000000
mean             0.000680            0.000680       0.000424       0.883993
std              0.000874            0.000874       0.000010       0.263156
min              0.000423            0.000423       0.000423       0.043478
25%              0.000423            0.000423       0.000423       1.000000
50%              0.000423            0.000423       0.000423       1.000000
75%              0.000423            0.000423       0.000423       1.000000
max              0.009738            0.009738       0.001270       1.000000

                lift       leverage    conviction
count  293284.000000  293284.000000  2.932840e+05
mean     1831.221971       0.000423           inf
std       795.980938       0.000010           NaN
min       102.695652       0.000419  1.045012e+00
25%      1181.000000       0.000423           inf
50%      2362.000000       0.000423           inf
75%      2362.000000       0.000423           inf
max      2362.000000       0.001265           inf
```

[168]:
```
assoc_rules = train_rules[(train_rules['lift'] >= 100) &␣
 ↪(train_rules['confidence'] >= 1)]
train_assoc_rules = assoc_rules.sort_values(by=['lift', 'support',␣
 ↪'confidence'], ascending = False)
```

[169]:
```
train_assoc_rules
```

[169]:
```
                                    antecedents    consequents  \
90                                (sku_2118790)   (sku_134727)
91                                 (sku_134727)  (sku_2118790)
```

210	(sku_7282946)	(sku_153077)
211	(sku_153077)	(sku_7282946)
214	(sku_158434)	(sku_4856105)
215	(sku_4856105)	(sku_158434)
404	(sku_3486768)	(sku_304816)
405	(sku_304816)	(sku_3486768)
598	(sku_5464851)	(sku_358142)
599	(sku_358142)	(sku_5464851)
600	(sku_5988914)	(sku_358142)
601	(sku_358142)	(sku_5988914)
604	(sku_8542489)	(sku_358142)
605	(sku_358142)	(sku_8542489)
612	(sku_362486)	(sku_3743023)
613	(sku_3743023)	(sku_362486)
614	(sku_6913238)	(sku_362486)
615	(sku_362486)	(sku_6913238)
712	(sku_433765)	(sku_7246549)
713	(sku_7246549)	(sku_433765)
850	(sku_503907)	(sku_7620930)
851	(sku_7620930)	(sku_503907)
882	(sku_524475)	(sku_1963776)
883	(sku_1963776)	(sku_524475)
886	(sku_524475)	(sku_5917683)
887	(sku_5917683)	(sku_524475)
888	(sku_9417053)	(sku_524475)
889	(sku_524475)	(sku_9417053)
968	(sku_581043)	(sku_3283740)
969	(sku_3283740)	(sku_581043)
...	...	...
205842	(sku_8983775, sku_9272339, sku_1889441, sku_35...	(sku_6246359)
207454	(sku_8983775, sku_3517442, sku_2631162, sku_23...	(sku_6246359)
207516	(sku_9272339, sku_3517442, sku_2631162, sku_23...	(sku_6246359)
207640	(sku_8983775, sku_9272339, sku_3517442, sku_26...	(sku_6246359)
207702	(sku_8983775, sku_9272339, sku_2631162, sku_23...	(sku_6246359)
207764	(sku_8983775, sku_9272339, sku_3517442, sku_23...	(sku_6246359)
211546	(sku_2616370, sku_8683742, sku_8831186, sku_48...	(sku_6246359)
211670	(sku_8983775, sku_9272339, sku_3517442, sku_26...	(sku_6246359)
212538	(sku_4156310, sku_3307427, sku_9343595, sku_54...	(sku_6246359)
212600	(sku_4156310, sku_9507892, sku_3307427, sku_54...	(sku_6246359)
212662	(sku_4156310, sku_9507892, sku_3307427, sku_93...	(sku_6246359)
212786	(sku_4156310, sku_9507892, sku_3307427, sku_93...	(sku_6246359)
212848	(sku_9507892, sku_3307427, sku_9343595, sku_54...	(sku_6246359)
213282	(sku_5373579, sku_4617567, sku_5441167, sku_34...	(sku_6246359)
213344	(sku_8983775, sku_5373579, sku_4617567, sku_54...	(sku_6246359)
213468	(sku_8983775, sku_5373579, sku_4617567, sku_34...	(sku_6246359)
213530	(sku_8983775, sku_4617567, sku_5441167, sku_34...	(sku_6246359)
213592	(sku_8983775, sku_5373579, sku_5441167, sku_34...	(sku_6246359)

```
215762  (sku_4156310, sku_9507892, sku_9343595, sku_54...  (sku_6246359)
216692  (sku_8983775, sku_5373579, sku_4617567, sku_54...  (sku_6246359)
261164  (sku_8983775, sku_1889441, sku_3517442, sku_26...  (sku_6246359)
261290  (sku_9272339, sku_1889441, sku_3517442, sku_26...  (sku_6246359)
261542  (sku_8983775, sku_9272339, sku_1889441, sku_35...  (sku_6246359)
261668  (sku_8983775, sku_9272339, sku_1889441, sku_26...  (sku_6246359)
261794  (sku_8983775, sku_9272339, sku_1889441, sku_35...  (sku_6246359)
261920  (sku_8983775, sku_9272339, sku_1889441, sku_35...  (sku_6246359)
262424  (sku_8983775, sku_9272339, sku_3517442, sku_26...  (sku_6246359)
264314  (sku_4156310, sku_9507892, sku_3307427, sku_93...  (sku_6246359)
264566  (sku_8983775, sku_5373579, sku_4617567, sku_54...  (sku_6246359)
289066  (sku_8983775, sku_9272339, sku_1889441, sku_35...  (sku_6246359)

        antecedent support  consequent support   support  confidence  \
90                0.000423            0.000423  0.000423         1.0
91                0.000423            0.000423  0.000423         1.0
210               0.000423            0.000423  0.000423         1.0
211               0.000423            0.000423  0.000423         1.0
214               0.000423            0.000423  0.000423         1.0
215               0.000423            0.000423  0.000423         1.0
404               0.000423            0.000423  0.000423         1.0
405               0.000423            0.000423  0.000423         1.0
598               0.000423            0.000423  0.000423         1.0
599               0.000423            0.000423  0.000423         1.0
600               0.000423            0.000423  0.000423         1.0
601               0.000423            0.000423  0.000423         1.0
604               0.000423            0.000423  0.000423         1.0
605               0.000423            0.000423  0.000423         1.0
612               0.000423            0.000423  0.000423         1.0
613               0.000423            0.000423  0.000423         1.0
614               0.000423            0.000423  0.000423         1.0
615               0.000423            0.000423  0.000423         1.0
712               0.000423            0.000423  0.000423         1.0
713               0.000423            0.000423  0.000423         1.0
850               0.000423            0.000423  0.000423         1.0
851               0.000423            0.000423  0.000423         1.0
882               0.000423            0.000423  0.000423         1.0
883               0.000423            0.000423  0.000423         1.0
886               0.000423            0.000423  0.000423         1.0
887               0.000423            0.000423  0.000423         1.0
888               0.000423            0.000423  0.000423         1.0
889               0.000423            0.000423  0.000423         1.0
968               0.000423            0.000423  0.000423         1.0
969               0.000423            0.000423  0.000423         1.0
...                    ...                 ...       ...         ...
205842            0.000423            0.009738  0.000423         1.0
207454            0.000423            0.009738  0.000423         1.0
```

207516	0.000423	0.009738	0.000423	1.0
207640	0.000423	0.009738	0.000423	1.0
207702	0.000423	0.009738	0.000423	1.0
207764	0.000423	0.009738	0.000423	1.0
211546	0.000423	0.009738	0.000423	1.0
211670	0.000423	0.009738	0.000423	1.0
212538	0.000423	0.009738	0.000423	1.0
212600	0.000423	0.009738	0.000423	1.0
212662	0.000423	0.009738	0.000423	1.0
212786	0.000423	0.009738	0.000423	1.0
212848	0.000423	0.009738	0.000423	1.0
213282	0.000423	0.009738	0.000423	1.0
213344	0.000423	0.009738	0.000423	1.0
213468	0.000423	0.009738	0.000423	1.0
213530	0.000423	0.009738	0.000423	1.0
213592	0.000423	0.009738	0.000423	1.0
215762	0.000423	0.009738	0.000423	1.0
216692	0.000423	0.009738	0.000423	1.0
261164	0.000423	0.009738	0.000423	1.0
261290	0.000423	0.009738	0.000423	1.0
261542	0.000423	0.009738	0.000423	1.0
261668	0.000423	0.009738	0.000423	1.0
261794	0.000423	0.009738	0.000423	1.0
261920	0.000423	0.009738	0.000423	1.0
262424	0.000423	0.009738	0.000423	1.0
264314	0.000423	0.009738	0.000423	1.0
264566	0.000423	0.009738	0.000423	1.0
289066	0.000423	0.009738	0.000423	1.0

	lift	leverage	conviction
90	2362.000000	0.000423	inf
91	2362.000000	0.000423	inf
210	2362.000000	0.000423	inf
211	2362.000000	0.000423	inf
214	2362.000000	0.000423	inf
215	2362.000000	0.000423	inf
404	2362.000000	0.000423	inf
405	2362.000000	0.000423	inf
598	2362.000000	0.000423	inf
599	2362.000000	0.000423	inf
600	2362.000000	0.000423	inf
601	2362.000000	0.000423	inf
604	2362.000000	0.000423	inf
605	2362.000000	0.000423	inf
612	2362.000000	0.000423	inf
613	2362.000000	0.000423	inf
614	2362.000000	0.000423	inf

```
615     2362.000000  0.000423         inf
712     2362.000000  0.000423         inf
713     2362.000000  0.000423         inf
850     2362.000000  0.000423         inf
851     2362.000000  0.000423         inf
882     2362.000000  0.000423         inf
883     2362.000000  0.000423         inf
886     2362.000000  0.000423         inf
887     2362.000000  0.000423         inf
888     2362.000000  0.000423         inf
889     2362.000000  0.000423         inf
968     2362.000000  0.000423         inf
969     2362.000000  0.000423         inf
...             ...       ...         ...
205842   102.695652  0.000419         inf
207454   102.695652  0.000419         inf
207516   102.695652  0.000419         inf
207640   102.695652  0.000419         inf
207702   102.695652  0.000419         inf
207764   102.695652  0.000419         inf
211546   102.695652  0.000419         inf
211670   102.695652  0.000419         inf
212538   102.695652  0.000419         inf
212600   102.695652  0.000419         inf
212662   102.695652  0.000419         inf
212786   102.695652  0.000419         inf
212848   102.695652  0.000419         inf
213282   102.695652  0.000419         inf
213344   102.695652  0.000419         inf
213468   102.695652  0.000419         inf
213530   102.695652  0.000419         inf
213592   102.695652  0.000419         inf
215762   102.695652  0.000419         inf
216692   102.695652  0.000419         inf
261164   102.695652  0.000419         inf
261290   102.695652  0.000419         inf
261542   102.695652  0.000419         inf
261668   102.695652  0.000419         inf
261794   102.695652  0.000419         inf
261920   102.695652  0.000419         inf
262424   102.695652  0.000419         inf
264314   102.695652  0.000419         inf
264566   102.695652  0.000419         inf
289066   102.695652  0.000419         inf

[243115 rows x 9 columns]
```

The process is now replicated for the testing dataset.

[170]:
```
%%capture

testing_data[testing_data >= 1] = 1
frequentItemsets_test = apriori(testing_data, min_support=0.00001,␣
 ↪use_colnames=True)
test_rules = association_rules(frequentItemsets_test, metric="lift",␣
 ↪min_threshold=1)
assoc_rules = test_rules[(test_rules['lift'] >= 100) &␣
 ↪(test_rules['confidence'] >= 1)]
test_assoc_rules = assoc_rules.sort_values(by=['lift', 'support',␣
 ↪'confidence'],ascending = False)
```

```
/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

[171]:
```
test_assoc_rules
```

[171]:

	antecedents	\
8	(sku_129494)	
9	(sku_433765)	
18	(sku_7754715)	
19	(sku_157625)	
28	(sku_8519573)	
29	(sku_297301)	
32	(sku_313920)	
33	(sku_9559575)	
66	(sku_464685)	
67	(sku_4328372)	
102	(sku_617092)	
103	(sku_3543767)	
118	(sku_9890223)	
119	(sku_808731)	
144	(sku_1124554)	
145	(sku_9007392)	
146	(sku_3779378)	
147	(sku_1349409)	
148	(sku_1368356)	
149	(sku_9918125)	
190	(sku_1978847)	
191	(sku_8237759)	
204	(sku_2061199)	
205	(sku_4128431)	

```
206                                    (sku_2061199)
207                                    (sku_6246359)
216                                    (sku_3497924)
217                                    (sku_2159229)
218                                    (sku_9491276)
219                                    (sku_2159229)
...                                             ...
470                                    (sku_8278844)
475                                    (sku_9760024)
612                                    (sku_7386821)
706                          (sku_172986, sku_6116307)
710                          (sku_9621612, sku_172986)
1202                        (sku_8278844, sku_1653173)
1269                        (sku_9621612, sku_2049781)
1478                        (sku_2603067, sku_5498692)
1484                        (sku_2603067, sku_7766405)
1490                        (sku_8911575, sku_2603067)
1524                        (sku_2878582, sku_6680056)
1526                        (sku_6680056, sku_3681414)
1532                        (sku_7766405, sku_5498692)
1538                        (sku_8911575, sku_5498692)
1543                        (sku_9621612, sku_6116307)
1550                        (sku_8911575, sku_7766405)
1566                        (sku_3508247, sku_3129701)
1668                        (sku_6493835, sku_3931575)
1674                        (sku_6887725, sku_3931575)
1681                        (sku_3951543, sku_4978283)
1686                        (sku_6887725, sku_6493835)
1956                        (sku_9407745, sku_7766605)
1975             (sku_9621612, sku_172986, sku_6116307)
3195            (sku_2603067, sku_7766405, sku_5498692)
3209            (sku_8911575, sku_2603067, sku_5498692)
3223            (sku_8911575, sku_2603067, sku_7766405)
3251            (sku_8911575, sku_7766405, sku_5498692)
3332            (sku_6887725, sku_6493835, sku_3931575)
3716                        (sku_1983514, sku_3998011)
4300  (sku_8911575, sku_2603067, sku_7766405, sku_54...

                               consequents  antecedent support  \
8                               (sku_433765)            0.001698
9                               (sku_129494)            0.001698
18                              (sku_157625)            0.001698
19                             (sku_7754715)            0.001698
28                              (sku_297301)            0.001698
29                             (sku_8519573)            0.001698
32                             (sku_9559575)            0.001698
33                              (sku_313920)            0.001698
```

66	(sku_4328372)	0.001698
67	(sku_464685)	0.001698
102	(sku_3543767)	0.001698
103	(sku_617092)	0.001698
118	(sku_808731)	0.001698
119	(sku_9890223)	0.001698
144	(sku_9007392)	0.001698
145	(sku_1124554)	0.001698
146	(sku_1349409)	0.001698
147	(sku_3779378)	0.001698
148	(sku_9918125)	0.001698
149	(sku_1368356)	0.001698
190	(sku_8237759)	0.001698
191	(sku_1978847)	0.001698
204	(sku_4128431)	0.001698
205	(sku_2061199)	0.001698
206	(sku_6246359)	0.001698
207	(sku_2061199)	0.001698
216	(sku_2159229)	0.001698
217	(sku_3497924)	0.001698
218	(sku_2159229)	0.001698
219	(sku_9491276)	0.001698
...	...	...
470	(sku_4706642)	0.001698
475	(sku_4706642)	0.001698
612	(sku_9936627)	0.001698
706	(sku_2878582)	0.001698
710	(sku_2878582)	0.001698
1202	(sku_4706642)	0.001698
1269	(sku_3681414)	0.001698
1478	(sku_2878582)	0.001698
1484	(sku_2878582)	0.001698
1490	(sku_2878582)	0.001698
1524	(sku_3681414)	0.001698
1526	(sku_2878582)	0.001698
1532	(sku_2878582)	0.001698
1538	(sku_2878582)	0.001698
1543	(sku_2878582)	0.001698
1550	(sku_2878582)	0.001698
1566	(sku_3681414)	0.001698
1668	(sku_3681414)	0.001698
1674	(sku_3681414)	0.001698
1681	(sku_3681414)	0.001698
1686	(sku_3681414)	0.001698
1956	(sku_9327745)	0.001698
1975	(sku_2878582)	0.001698
3195	(sku_2878582)	0.001698

3209	(sku_2878582)	0.001698
3223	(sku_2878582)	0.001698
3251	(sku_2878582)	0.001698
3332	(sku_3681414)	0.001698
3716	(sku_2416897, sku_368506, sku_9297426)	0.001698
4300	(sku_2878582)	0.001698

	consequent support	support	confidence	lift	leverage	conviction
8	0.001698	0.001698	1.0	589.0	0.001695	inf
9	0.001698	0.001698	1.0	589.0	0.001695	inf
18	0.001698	0.001698	1.0	589.0	0.001695	inf
19	0.001698	0.001698	1.0	589.0	0.001695	inf
28	0.001698	0.001698	1.0	589.0	0.001695	inf
29	0.001698	0.001698	1.0	589.0	0.001695	inf
32	0.001698	0.001698	1.0	589.0	0.001695	inf
33	0.001698	0.001698	1.0	589.0	0.001695	inf
66	0.001698	0.001698	1.0	589.0	0.001695	inf
67	0.001698	0.001698	1.0	589.0	0.001695	inf
102	0.001698	0.001698	1.0	589.0	0.001695	inf
103	0.001698	0.001698	1.0	589.0	0.001695	inf
118	0.001698	0.001698	1.0	589.0	0.001695	inf
119	0.001698	0.001698	1.0	589.0	0.001695	inf
144	0.001698	0.001698	1.0	589.0	0.001695	inf
145	0.001698	0.001698	1.0	589.0	0.001695	inf
146	0.001698	0.001698	1.0	589.0	0.001695	inf
147	0.001698	0.001698	1.0	589.0	0.001695	inf
148	0.001698	0.001698	1.0	589.0	0.001695	inf
149	0.001698	0.001698	1.0	589.0	0.001695	inf
190	0.001698	0.001698	1.0	589.0	0.001695	inf
191	0.001698	0.001698	1.0	589.0	0.001695	inf
204	0.001698	0.001698	1.0	589.0	0.001695	inf
205	0.001698	0.001698	1.0	589.0	0.001695	inf
206	0.001698	0.001698	1.0	589.0	0.001695	inf
207	0.001698	0.001698	1.0	589.0	0.001695	inf
216	0.001698	0.001698	1.0	589.0	0.001695	inf
217	0.001698	0.001698	1.0	589.0	0.001695	inf
218	0.001698	0.001698	1.0	589.0	0.001695	inf
219	0.001698	0.001698	1.0	589.0	0.001695	inf
...	...	...	...	...	...	...
470	0.008489	0.001698	1.0	117.8	0.001683	inf
475	0.008489	0.001698	1.0	117.8	0.001683	inf
612	0.008489	0.001698	1.0	117.8	0.001683	inf
706	0.008489	0.001698	1.0	117.8	0.001683	inf
710	0.008489	0.001698	1.0	117.8	0.001683	inf
1202	0.008489	0.001698	1.0	117.8	0.001683	inf
1269	0.008489	0.001698	1.0	117.8	0.001683	inf
1478	0.008489	0.001698	1.0	117.8	0.001683	inf

```
1484          0.008489  0.001698         1.0  117.8  0.001683         inf
1490          0.008489  0.001698         1.0  117.8  0.001683         inf
1524          0.008489  0.001698         1.0  117.8  0.001683         inf
1526          0.008489  0.001698         1.0  117.8  0.001683         inf
1532          0.008489  0.001698         1.0  117.8  0.001683         inf
1538          0.008489  0.001698         1.0  117.8  0.001683         inf
1543          0.008489  0.001698         1.0  117.8  0.001683         inf
1550          0.008489  0.001698         1.0  117.8  0.001683         inf
1566          0.008489  0.001698         1.0  117.8  0.001683         inf
1668          0.008489  0.001698         1.0  117.8  0.001683         inf
1674          0.008489  0.001698         1.0  117.8  0.001683         inf
1681          0.008489  0.001698         1.0  117.8  0.001683         inf
1686          0.008489  0.001698         1.0  117.8  0.001683         inf
1956          0.008489  0.001698         1.0  117.8  0.001683         inf
1975          0.008489  0.001698         1.0  117.8  0.001683         inf
3195          0.008489  0.001698         1.0  117.8  0.001683         inf
3209          0.008489  0.001698         1.0  117.8  0.001683         inf
3223          0.008489  0.001698         1.0  117.8  0.001683         inf
3251          0.008489  0.001698         1.0  117.8  0.001683         inf
3332          0.008489  0.001698         1.0  117.8  0.001683         inf
3716          0.008489  0.001698         1.0  117.8  0.001683         inf
4300          0.008489  0.001698         1.0  117.8  0.001683         inf

[1070 rows x 9 columns]
```

To compare how well the training set works against the test set, I determined whether similar SKU associations were created in the test set. Unfortunately, only 2 similar SKU associations were found. One of them is already included in the top 100 SKU associations, so the only additional one that needs to be added is the one with the index larger than 100.

[172]:
```
combined_results_train = []
combined_results_test = []

validated = []

for i in range(train_assoc_rules.shape[0]):
    combined_results_train.append(str(train_assoc_rules.iloc[i,0]) +␣
 ↪str(train_assoc_rules.iloc[i,1]))

for i in range(test_assoc_rules.shape[0]):
    combined_results_test.append(str(test_assoc_rules.iloc[i,0]) +␣
 ↪str(test_assoc_rules.iloc[i,1]))

for i in range(test_assoc_rules.shape[0]):
    if combined_results_test[i] in combined_results_train:
        validated.append(i)
```

[173]:
```
validated
```

[173]:
```
[51, 784]
```

[178]:
```
val_1 = combined_results_train.index(combined_results_test[784])

final_SKU = train_assoc_rules.iloc[[val_1],:]
```

A final SKU dataframe was created by adding the top 98 SKU associations from the training dataset and adding the remaining two SKU associations from the validation set. This dataset is ordered by lift, support and confidence.

[179]:
```
remaining_98 = train_assoc_rules.iloc[:99,:]
final_SKU = pd.concat([final_SKU, remaining_98])
```

[180]:
```
final_SKU.sort_values(['lift', 'support','confidence'], ascending = False)
```

[180]:

	antecedents	consequents	antecedent support	consequent support	\
90	(sku_2118790)	(sku_134727)	0.000423	0.000423	
91	(sku_134727)	(sku_2118790)	0.000423	0.000423	
210	(sku_7282946)	(sku_153077)	0.000423	0.000423	
211	(sku_153077)	(sku_7282946)	0.000423	0.000423	
214	(sku_158434)	(sku_4856105)	0.000423	0.000423	
215	(sku_4856105)	(sku_158434)	0.000423	0.000423	
404	(sku_3486768)	(sku_304816)	0.000423	0.000423	
405	(sku_304816)	(sku_3486768)	0.000423	0.000423	
598	(sku_5464851)	(sku_358142)	0.000423	0.000423	
599	(sku_358142)	(sku_5464851)	0.000423	0.000423	
600	(sku_5988914)	(sku_358142)	0.000423	0.000423	
601	(sku_358142)	(sku_5988914)	0.000423	0.000423	
604	(sku_8542489)	(sku_358142)	0.000423	0.000423	
605	(sku_358142)	(sku_8542489)	0.000423	0.000423	
612	(sku_362486)	(sku_3743023)	0.000423	0.000423	
613	(sku_3743023)	(sku_362486)	0.000423	0.000423	
614	(sku_6913238)	(sku_362486)	0.000423	0.000423	
615	(sku_362486)	(sku_6913238)	0.000423	0.000423	
712	(sku_433765)	(sku_7246549)	0.000423	0.000423	
713	(sku_7246549)	(sku_433765)	0.000423	0.000423	
850	(sku_503907)	(sku_7620930)	0.000423	0.000423	
851	(sku_7620930)	(sku_503907)	0.000423	0.000423	
882	(sku_524475)	(sku_1963776)	0.000423	0.000423	
883	(sku_1963776)	(sku_524475)	0.000423	0.000423	
886	(sku_524475)	(sku_5917683)	0.000423	0.000423	
887	(sku_5917683)	(sku_524475)	0.000423	0.000423	
888	(sku_9417053)	(sku_524475)	0.000423	0.000423	
889	(sku_524475)	(sku_9417053)	0.000423	0.000423	
968	(sku_581043)	(sku_3283740)	0.000423	0.000423	
969	(sku_3283740)	(sku_581043)	0.000423	0.000423	
...	...	...	...	...	
2578	(sku_9049654)	(sku_1617258)	0.000423	0.000423	
2579	(sku_1617258)	(sku_9049654)	0.000423	0.000423	
2676	(sku_9907725)	(sku_1652352)	0.000423	0.000423	

2677	(sku_1652352)	(sku_9907725)	0.000423	0.000423
2870	(sku_1726593)	(sku_8928206)	0.000423	0.000423
2871	(sku_8928206)	(sku_1726593)	0.000423	0.000423
2878	(sku_1746864)	(sku_6189903)	0.000423	0.000423
2879	(sku_6189903)	(sku_1746864)	0.000423	0.000423
2900	(sku_1764166)	(sku_2731751)	0.000423	0.000423
2901	(sku_2731751)	(sku_1764166)	0.000423	0.000423
2902	(sku_3694213)	(sku_1764166)	0.000423	0.000423
2903	(sku_1764166)	(sku_3694213)	0.000423	0.000423
2950	(sku_1788856)	(sku_9388194)	0.000423	0.000423
2951	(sku_9388194)	(sku_1788856)	0.000423	0.000423
2956	(sku_1796791)	(sku_4467918)	0.000423	0.000423
2957	(sku_4467918)	(sku_1796791)	0.000423	0.000423
2988	(sku_2581489)	(sku_1854133)	0.000423	0.000423
2989	(sku_1854133)	(sku_2581489)	0.000423	0.000423
3120	(sku_1916600)	(sku_9826609)	0.000423	0.000423
3121	(sku_9826609)	(sku_1916600)	0.000423	0.000423
3184	(sku_4080106)	(sku_1941912)	0.000423	0.000423
3185	(sku_1941912)	(sku_4080106)	0.000423	0.000423
3234	(sku_1963776)	(sku_5917683)	0.000423	0.000423
3235	(sku_5917683)	(sku_1963776)	0.000423	0.000423
3236	(sku_9417053)	(sku_1963776)	0.000423	0.000423
3237	(sku_1963776)	(sku_9417053)	0.000423	0.000423
3362	(sku_2057800)	(sku_8000063)	0.000423	0.000423
3363	(sku_8000063)	(sku_2057800)	0.000423	0.000423
3368	(sku_2061791)	(sku_3053923)	0.000423	0.000423
7595	(sku_9706694)	(sku_6137996)	0.000423	0.001693

	support	confidence	lift	leverage	conviction
90	0.000423	1.0	2362.0	0.000423	inf
91	0.000423	1.0	2362.0	0.000423	inf
210	0.000423	1.0	2362.0	0.000423	inf
211	0.000423	1.0	2362.0	0.000423	inf
214	0.000423	1.0	2362.0	0.000423	inf
215	0.000423	1.0	2362.0	0.000423	inf
404	0.000423	1.0	2362.0	0.000423	inf
405	0.000423	1.0	2362.0	0.000423	inf
598	0.000423	1.0	2362.0	0.000423	inf
599	0.000423	1.0	2362.0	0.000423	inf
600	0.000423	1.0	2362.0	0.000423	inf
601	0.000423	1.0	2362.0	0.000423	inf
604	0.000423	1.0	2362.0	0.000423	inf
605	0.000423	1.0	2362.0	0.000423	inf
612	0.000423	1.0	2362.0	0.000423	inf
613	0.000423	1.0	2362.0	0.000423	inf
614	0.000423	1.0	2362.0	0.000423	inf
615	0.000423	1.0	2362.0	0.000423	inf

```
712   0.000423         1.0  2362.0  0.000423         inf
713   0.000423         1.0  2362.0  0.000423         inf
850   0.000423         1.0  2362.0  0.000423         inf
851   0.000423         1.0  2362.0  0.000423         inf
882   0.000423         1.0  2362.0  0.000423         inf
883   0.000423         1.0  2362.0  0.000423         inf
886   0.000423         1.0  2362.0  0.000423         inf
887   0.000423         1.0  2362.0  0.000423         inf
888   0.000423         1.0  2362.0  0.000423         inf
889   0.000423         1.0  2362.0  0.000423         inf
968   0.000423         1.0  2362.0  0.000423         inf
969   0.000423         1.0  2362.0  0.000423         inf
...        ...         ...     ...       ...         ...
2578  0.000423         1.0  2362.0  0.000423         inf
2579  0.000423         1.0  2362.0  0.000423         inf
2676  0.000423         1.0  2362.0  0.000423         inf
2677  0.000423         1.0  2362.0  0.000423         inf
2870  0.000423         1.0  2362.0  0.000423         inf
2871  0.000423         1.0  2362.0  0.000423         inf
2878  0.000423         1.0  2362.0  0.000423         inf
2879  0.000423         1.0  2362.0  0.000423         inf
2900  0.000423         1.0  2362.0  0.000423         inf
2901  0.000423         1.0  2362.0  0.000423         inf
2902  0.000423         1.0  2362.0  0.000423         inf
2903  0.000423         1.0  2362.0  0.000423         inf
2950  0.000423         1.0  2362.0  0.000423         inf
2951  0.000423         1.0  2362.0  0.000423         inf
2956  0.000423         1.0  2362.0  0.000423         inf
2957  0.000423         1.0  2362.0  0.000423         inf
2988  0.000423         1.0  2362.0  0.000423         inf
2989  0.000423         1.0  2362.0  0.000423         inf
3120  0.000423         1.0  2362.0  0.000423         inf
3121  0.000423         1.0  2362.0  0.000423         inf
3184  0.000423         1.0  2362.0  0.000423         inf
3185  0.000423         1.0  2362.0  0.000423         inf
3234  0.000423         1.0  2362.0  0.000423         inf
3235  0.000423         1.0  2362.0  0.000423         inf
3236  0.000423         1.0  2362.0  0.000423         inf
3237  0.000423         1.0  2362.0  0.000423         inf
3362  0.000423         1.0  2362.0  0.000423         inf
3363  0.000423         1.0  2362.0  0.000423         inf
3368  0.000423         1.0  2362.0  0.000423         inf
7595  0.000423         1.0   590.5  0.000423         inf

[100 rows x 9 columns]
```

0.5 Conclusion

The above dataframe represents the 100 SKU associations present in the 1000 SKUs selected earlier. Hence, by moving products with these SKUs closer to each other, there could be a greater number of purchases. Alternatively, if these products are further away from each other, customers would be forced to walk across the store and may be tempted to buy additional products. Essentially, Dillards can use this information to effectively move around their products to affect customer revenue.