lab 09 - Frequent Patterns

Name: Robb Alexander and Ryan Bailis Class: CSCI349 Semester: 2021SP Instructor: Brian King

In [2]:
```
# Setting things up
import numpy as np
import pandas as pd
import plotly.express as px
```

1) Add the import statements

In [3]:
```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

2) Go to the page: http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/ (http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/) . Enter the list dataset shown on the page. Then, copy the example code that transforms the list to a numpy encoded array, then to a pandas DataFrame with the correct column names. Output your data frame.

In [4]:
```
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'
],['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],['Milk',
'Apple', 'Kidney Beans', 'Eggs'],['Milk', 'Unicorn', 'Corn', 'Kidney Bea
ns', 'Yogurt'],['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream',
'Eggs']]

te = TransactionEncoder()
te_ary = te.fit_transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

Out[4]:

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

3) Show the result of describe() and info() on your dataframe.

In [5]:
```
df.describe()
```

Out[5]:

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
count	5	5	5	5	5	5	5	5	5	5	5
unique	2	2	2	2	2	1	2	2	2	2	2
top	False	False	False	True	False	True	True	False	True	False	True
freq	4	3	4	4	4	5	3	3	3	4	3

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Apple         5 non-null      bool
 1   Corn          5 non-null      bool
 2   Dill          5 non-null      bool
 3   Eggs          5 non-null      bool
 4   Ice cream     5 non-null      bool
 5   Kidney Beans  5 non-null      bool
 6   Milk          5 non-null      bool
 7   Nutmeg        5 non-null      bool
 8   Onion         5 non-null      bool
 9   Unicorn       5 non-null      bool
 10  Yogurt        5 non-null      bool
dtypes: bool(11)
memory usage: 183.0 bytes
```

4) Following along the mlxtend user guide, use the apriori algorithm to final all frequent itemsets with a min_support of 0.6. Show the resulting dataframe, and store the result, since you'll have many selection exercises next. All selection exercises must be done from this resulting frame. Set use_colnames=True. It'll be much easier to interpret your patterns.

In [7]:
```
df = apriori(df, min_support=0.6, use_colnames=True)
df
```

Out[7]:

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
2	0.6	(Milk)
3	0.6	(Onion)
4	0.6	(Yogurt)
5	0.8	(Kidney Beans, Eggs)
6	0.6	(Eggs, Onion)
7	0.6	(Kidney Beans, Milk)
8	0.6	(Kidney Beans, Onion)
9	0.6	(Kidney Beans, Yogurt)
10	0.6	(Kidney Beans, Eggs, Onion)

5) Select all frequent itemsets that have support => 0.8

In [8]:
```
df[df.support >= 0.8]
```

Out[8]:

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
5	0.8	(Kidney Beans, Eggs)

6) Select all frequent itemsets with at least 2 items. In their documentation, they often create additional helper variables to make it easier to select your data. That's entirely up to you. (I tend to be a purist, and reserve additional variables for only very complex selection criteria. Computing the length of an itemset is not one of them! It's entirely up to you. Do what makes the most sense to you.)

In [9]:
```
df[df.itemsets.apply(len) >= 2]
```

Out[9]:

	support	itemsets
5	0.8	(Kidney Beans, Eggs)
6	0.6	(Eggs, Onion)
7	0.6	(Kidney Beans, Milk)
8	0.6	(Kidney Beans, Onion)
9	0.6	(Kidney Beans, Yogurt)
10	0.6	(Kidney Beans, Eggs, Onion)

7) Select the frequent itemsets that contain an 'Onion' in the itemset.

In [10]:
```
df[df.itemsets.apply(lambda x: "Onion" in x)]
```

Out[10]:

	support	itemsets
3	0.6	(Onion)
6	0.6	(Eggs, Onion)
8	0.6	(Kidney Beans, Onion)
10	0.6	(Kidney Beans, Eggs, Onion)

8) Select the frequent itemsets that contain both 'Onion' and 'Eggs' in the itemset. (HINT: You should have 2 frequent itemsets selected. And, if you haven't learned about the set type in Python and all of the standard set operations, they can really make these types of questions much easier.)

In [11]:
```
df[df.itemsets.apply(lambda x: "Onion" in x and "Eggs" in x)]
```

Out[11]:

	support	itemsets
6	0.6	(Eggs, Onion)
10	0.6	(Kidney Beans, Eggs, Onion)

9) Select the frequent itemsets that contain either an 'Onion' or 'Kidney Beans' (or both) in the itemset. (HINT: You should have 8 frequent itemsets output.)

In [12]:
```
df[df.itemsets.apply(lambda x: "Onion" in x or "Kidney Beans" in x)]
```

Out[12]:

	support	itemsets
1	1.0	(Kidney Beans)
3	0.6	(Onion)
5	0.8	(Kidney Beans, Eggs)
6	0.6	(Eggs, Onion)
7	0.6	(Kidney Beans, Milk)
8	0.6	(Kidney Beans, Onion)
9	0.6	(Kidney Beans, Yogurt)
10	0.6	(Kidney Beans, Eggs, Onion)

10) http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/ (http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/) contains the all the information and guidance you need for working with association rules from frequent patterns. (NOTE – they mention a function called generate_rules(). The function is association_rules().) Generate rules with a minimum confidence of 0.7. Store your resulting data frame called rules. Show the entire data frame. (You should have 12 rules).

In [13]:
```
rules = association_rules(df, metric="confidence", min_threshold=0.7)
rules
```

Out[13]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	cor
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.00	
1	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.00	
2	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	
3	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	
4	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	
5	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	
6	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	
7	(Kidney Beans, Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	
8	(Kidney Beans, Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	
9	(Eggs, Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	
10	(Eggs)	(Kidney Beans, Onion)	0.8	0.6	0.6	0.75	1.25	0.12	
11	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	

11) Output the top 5 rules in descending order by "lift", with the secondary sort key by "confidence".

In [14]:
```
rules.sort_values(by=["lift", "confidence"], ascending=False)[:5]
```

Out[14]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	cor
3	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	
8	(Kidney Beans, Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	
11	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	
2	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	
7	(Kidney Beans, Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	

12) Select all rules that have a 1.0 support for the antecedent.

In [15]:
```
rules[rules["antecedent support"] == 1]
```

Out[15]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	convi
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.8	1.0	0.0	

13) Select all rules that have at least 3 or more items represented in the rule (i.e. the union of the antecedent and consequent >= 3.)

In [16]:
```
rules[rules.apply(lambda x: len(x.antecedents) + len(x.consequents) >= 3
, axis=1)]
```

Out[16]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	cor
7	(Kidney Beans, Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	
8	(Kidney Beans, Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	
9	(Eggs, Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	
10	(Eggs)	(Kidney Beans, Onion)	0.8	0.6	0.6	0.75	1.25	0.12	
11	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	

14) Select the rules that have confidence >= 0.75 and a lift > 1

In [17]:
```
rules[(rules.confidence >= 0.75) & (rules.lift > 1)]
```

Out[17]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	cor
3	(Onion)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	
8	(Kidney Beans, Onion)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	
11	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	

15) Read in the Chipotle dataset: url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv (https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv)' df_chip = pd.read_csv(url, sep = '\t')

In [18]:
```
url = "https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv"
df_chip = pd.read_csv(url, sep="\t")
```

16) Show the result of df_chip.info(verbose=True) You should have five variables.

In [19]:
```
df_chip.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4622 entries, 0 to 4621
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   order_id            4622 non-null   int64 
 1   quantity            4622 non-null   int64 
 2   item_name           4622 non-null   object
 3   choice_description  3376 non-null   object
 4   item_price          4622 non-null   object
dtypes: int64(2), object(3)
memory usage: 180.7+ KB
```

17) Show the result of describe(include='all')

In [20]:
```
df_chip.describe(include="all")
```

Out[20]:

	order_id	quantity	item_name	choice_description	item_price
count	4622.000000	4622.000000	4622	3376	4622
unique	NaN	NaN	50	1043	78
top	NaN	NaN	Chicken Bowl	[Diet Coke]	$8.75
freq	NaN	NaN	726	134	730
mean	927.254868	1.075725	NaN	NaN	NaN
std	528.890796	0.410186	NaN	NaN	NaN
min	1.000000	1.000000	NaN	NaN	NaN
25%	477.250000	1.000000	NaN	NaN	NaN
50%	926.000000	1.000000	NaN	NaN	NaN
75%	1393.000000	1.000000	NaN	NaN	NaN
max	1834.000000	15.000000	NaN	NaN	NaN

18) Show the first 10 observations

In [21]:
```
df_chip.head(10)
```

Out[21]:

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	$10.98
6	3	1	Side of Chips	NaN	$1.69
7	4	1	Steak Burrito	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	$11.75
8	4	1	Steak Soft Tacos	[Tomatillo Green Chili Salsa, [Pinto Beans, Ch...	$9.25
9	5	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Pinto...	$9.25

19) Study what you see so far. Minimally, you should notice that you have three variables that need to be transformed into usable types. Which ones, and what do you need to do with them?

The three are the item_names choice_descriptions and item_prices. This is because they are currently string representations.
We can change the item prices to a float, which is simple. We can use categoricals to classify each item name into an int. Finally, we can make the choice descriptions into a list of ints by also making the subportions into categoricals.

20) Let's start doing some preprocessing. Convert the item_price field to a floating-point number, and then show the result of df_chip.item_price.describe() to show that it is indeed a numeric variable now.

In [22]:
```
df_chip.item_price = df_chip.item_price.map(lambda x: x.replace(' ', '')
.replace(',', '').replace('$', ''))
df_chip.item_price = pd.to_numeric(df_chip.item_price)
df_chip.item_price.describe()
```

Out[22]:
```
count    4622.000000
mean        7.464336
std         4.245557
min         1.090000
25%         3.390000
50%         8.750000
75%         9.250000
max        44.250000
Name: item_price, dtype: float64
```

21) Now, convert the item_name to a categorical variable.

In [23]:
```
df_chip.item_name = pd.Categorical(df_chip.item_name)
df_chip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4622 entries, 0 to 4621
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   order_id            4622 non-null   int64
 1   quantity            4622 non-null   int64
 2   item_name           4622 non-null   category
 3   choice_description  3376 non-null   object
 4   item_price          4622 non-null   float64
dtypes: category(1), float64(1), int64(2), object(1)
memory usage: 151.5+ KB
```

22) How many unique item_name values are there?

In [24]:
```
len(df_chip.item_name.unique())
```

Out[24]: 50

23) Show all of the unique values in item_name . Do you see any potential problems? (Leave them! Don't fix them. Just pay close attention. Need a hint? Salsa)

In [25]:

```
for i in df_chip.item_name.unique():
    print(i)

# Chips and Tomatillo-Green Chili Salsa
# Chips and Tomatillo Green Chili Salsa
# Chips and Tomatillo Red Chili Salsa
# Chips and Tomatillo-Red Chili Salsa
# Chips and Roasted Chili-Corn Salsa
# Chips and Roasted Chili Corn Salsa
# Chips and Mild Fresh Tomato Salsa
# Chips and Fresh Tomato Salsa
```

```
Chips and Fresh Tomato Salsa
Izze
Nantucket Nectar
Chips and Tomatillo-Green Chili Salsa
Chicken Bowl
Side of Chips
Steak Burrito
Steak Soft Tacos
Chips and Guacamole
Chicken Crispy Tacos
Chicken Soft Tacos
Chicken Burrito
Canned Soda
Barbacoa Burrito
Carnitas Burrito
Carnitas Bowl
Bottled Water
Chips and Tomatillo Green Chili Salsa
Barbacoa Bowl
Chips
Chicken Salad Bowl
Steak Bowl
Barbacoa Soft Tacos
Veggie Burrito
Veggie Bowl
Steak Crispy Tacos
Chips and Tomatillo Red Chili Salsa
Barbacoa Crispy Tacos
Veggie Salad Bowl
Chips and Roasted Chili-Corn Salsa
Chips and Roasted Chili Corn Salsa
Carnitas Soft Tacos
Chicken Salad
Canned Soft Drink
Steak Salad Bowl
6 Pack Soft Drink
Chips and Tomatillo-Red Chili Salsa
Bowl
Burrito
Crispy Tacos
Carnitas Crispy Tacos
Steak Salad
Chips and Mild Fresh Tomato Salsa
Veggie Soft Tacos
Carnitas Salad Bowl
Barbacoa Salad Bowl
Salad
Veggie Crispy Tacos
Veggie Salad
Carnitas Salad
```

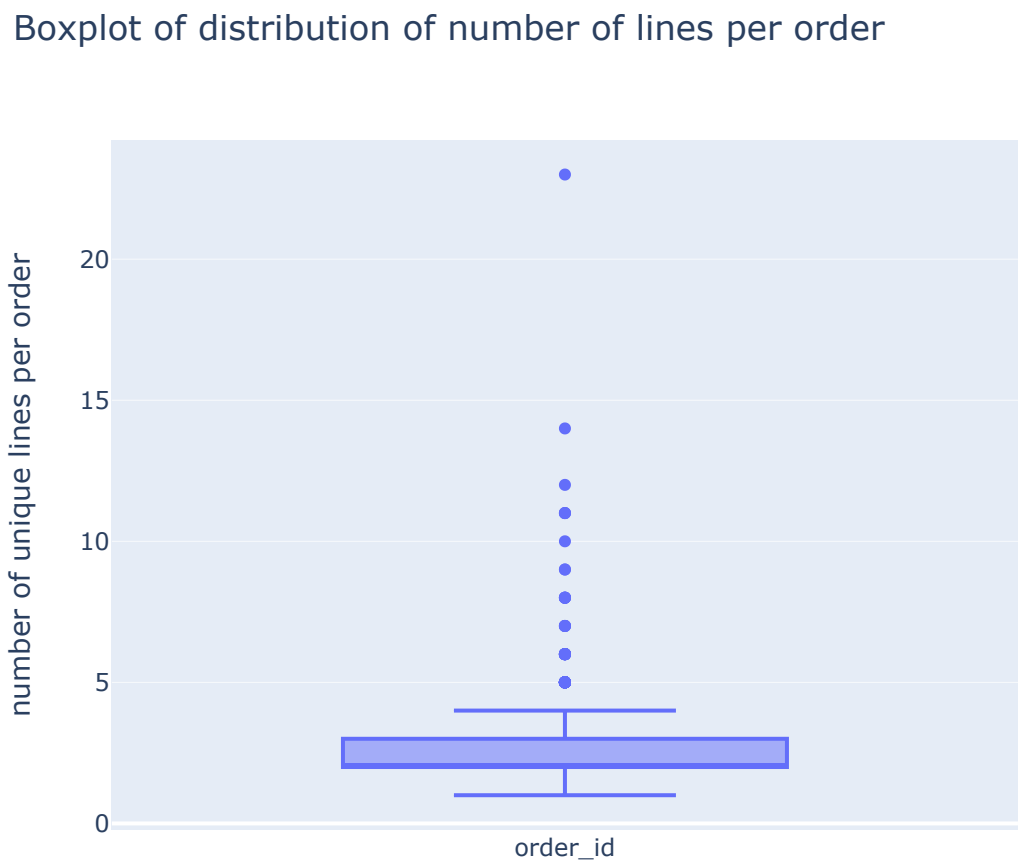
24) How many distinct orders are there?

```
In [26]: len(df_chip.order_id.unique())
```

```
Out[26]: 1834
```

25) Show a boxplot of the number of line items per order. Do NOT consider the quantity of each item, just the total count of line items. (NOTE: Many items appear multiple times in an order. Don't worry about that. Just count the number of items per order.) Then, comment on the distribution of the total number of items per order.

```
In [27]: px.box(df_chip.order_id.value_counts(), labels={"variable":"", "value":
         "number of unique lines per order"}, title="Boxplot of distribution of n
         umber of lines per order")
```



26) Show a boxplot of the TOTAL number of items per order. Now, you must consider the quantity of each item in each order. (The box plot will be similar to the previous, with the exception that there will be more outliers, and the maximum outlier will be 35.)

In [28]:
```
order_counts = df_chip.groupby("order_id").quantity.sum()
px.box(order_counts, labels={"variable":"", "value":"total itesm per ord
er"}, title="Boxplot of distribution of total items per order")
```

27) What were the top 5 ordered items by total quantity? Report the item and its total quantity ordered. Be sure to consider the quantity of each item ordered!

In [29]:
```
total_quantity = df_chip.groupby("item_name").aggregate(np.sum).quantity
total_quantity.sort_values(ascending=False)[:5]
```

Out[29]:
```
item_name
Chicken Bowl           761
Chicken Burrito        591
Chips and Guacamole    506
Steak Burrito          386
Canned Soft Drink      351
Name: quantity, dtype: int64
```

28) What is the total number of "Steak Burrito" ordered?

In [30]:
```
total_quantity["Steak Burrito"]
```

Out[30]:
```
386
```

29) What is mean price for an order (NOTE – This is NOT just a mean of the item_price column!)

In [31]:
```
df_chip.item_price.mean()
df_chip.groupby("order_id").item_price.sum().mean()
```

Out[31]:
```
18.811428571428568
```

30) What was total revenue for the day?

In [32]:
```
df_chip.item_price.sum()
```

Out[32]:
```
34500.16
```

31) What was the largest total price for a single order? Show the order number and the total price.

In [33]:
```
largest = df_chip.groupby("order_id").item_price.sum()
print("Order number", largest.argmax() + 1, "costing $" + str(largest.ma
x()))
```

```
Order number 926 costing $205.25
```

32) Show the entire order to your answer to the previous question (NOTE: This should show you how some orders can contain multiple lines of the same item. Not uncommon!)

In [34]:
```
indexes = df_chip.groupby("order_id").item_price.groups[largest.argmax()
+ 1]
df_chip.iloc[indexes]
```

Out[34]:

	order_id	quantity	item_name	choice_description	item_price
2304	926	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Sour Cream, Lettuce]]	9.25
2305	926	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Fajita Vegetables,...	8.75
2306	926	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Fajita Vegetables,...	8.75
2307	926	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Fajita Vegetables,...	8.75
2308	926	1	Steak Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Lettu...	9.25
2309	926	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	8.75
2310	926	1	Steak Burrito	[Roasted Chili Corn Salsa, [Rice, Cheese, Sour...	9.25
2311	926	1	Chicken Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	8.75
2312	926	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Lettuce]]	8.75
2313	926	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Cheese, Sour Cream...	8.75
2314	926	1	Chicken Salad Bowl	[Roasted Chili Corn Salsa, [Rice, Sour Cream]]	8.75
2315	926	1	Steak Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	9.25
2316	926	1	Chicken Burrito	[Roasted Chili Corn Salsa, [Rice, Black Beans,...	8.75
2317	926	1	Steak Bowl	[Roasted Chili Corn Salsa, [Rice, Black Beans,...	9.25
2318	926	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Fajita Vegetables,...	8.75
2319	926	1	Steak Bowl	[Fresh Tomato Salsa, [Rice, Cheese]]	9.25
2320	926	1	Chicken Burrito	[Fresh Tomato Salsa, [Rice, Cheese, Lettuce]]	8.75
2321	926	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Rice, Pinto Beans,...	8.75
2322	926	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Rice, Cheese]]	8.75
2323	926	1	Barbacoa Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	9.25
2324	926	1	Chicken Burrito	[Tomatillo Red Chili Salsa, [Rice, Cheese, Sou...	8.75
2325	926	1	Steak Bowl	[Tomatillo Red Chili Salsa, [Rice, Black Beans...	9.25
2326	926	1	Veggie Bowl	[Roasted Chili Corn Salsa, [Rice, Black Beans,...	8.75

33) What order had the largest total quantity of items purchased? Show the order number and the total number of items

In [35]:
```
largest = df_chip.groupby("order_id").quantity.sum()
print("Order number", largest.argmax() + 1, "with", largest.max(), "item
s purchased")
```

```
Order number 1443 with 35 items purchased
```

34) Show the entire order to your answer to the previous question

In [36]:
```
indexes = df_chip.groupby("order_id").quantity.groups[largest.argmax() +
1]
df_chip.iloc[indexes]
```

Out[36]:

	order_id	quantity	item_name	choice_description	item_price
3598	1443	15	Chips and Fresh Tomato Salsa	NaN	44.25
3599	1443	7	Bottled Water	NaN	10.50
3600	1443	1	6 Pack Soft Drink	[Coke]	6.49
3601	1443	3	Veggie Burrito	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	33.75
3602	1443	4	Chicken Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	35.00
3603	1443	3	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	27.75
3604	1443	2	Bottled Water	NaN	3.00

35) Recall that this dataset is a set of transactions, where each observation represents one item purchased as part of an order_id. However, the data are not read in this way. You need to transform this dataset to a collection of binary encoded transactions, where each row represents ONE transaction, and the columns are binary encoded variables, with each variable representing ONE item available for purchase at Chipotle. Convert your data. Your resulting data frame should have an index representing the order_id, and columns representing each possible item from the item_name variable. For now, a transaction will ignore the quantity of item purchased. The shape of your resulting data frame should be (1834,50)

In [37]:
```
combined_df = df_chip.groupby("order_id").item_name.unique()
combined_df = combined_df.apply(list)

te_ary = te.fit_transform(combined_df)
df_enc = pd.DataFrame(te_ary, columns=te.columns_, index=df_chip.groupby
("order_id").count().index)
```

36) Show the first 10 observations from your transaction data

In [38]:
```
df_enc.head(10)
```

Out[38]:

order_id	6 Pack Soft Drink	Barbacoa Bowl	Barbacoa Burrito	Barbacoa Crispy Tacos	Barbacoa Salad Bowl	Barbacoa Soft Tacos	Bottled Water	Bowl	Burrito	C
1	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	
5	False	False	False	False	False	False	False	False	False	
6	False	False	False	False	False	False	False	False	False	
7	False	False	False	False	False	False	False	False	False	
8	False	False	False	False	False	False	False	False	False	
9	False	False	False	False	False	False	False	False	False	
10	False	False	False	False	False	False	False	False	False	

10 rows × 50 columns

37) Too often, many start by considering a minimum support that is arbitrarily large. Go ahead and use the apriori method to generate frequent itemsets with a minsup value of 0.5. What happened?

In [39]:
```
"""
No outputs with supports over 0.5
"""
df_apri = apriori(df_enc, min_support=0.5, use_colnames=True)
df_apri
```

Out[39]:

	support	itemsets

38) Take a step back. Your previous outcome is why you ALWAYS perform essential EDA tasks before you dive into mining a dataset! Report a table that shows the number of transactions each item occurred in, sorted in order of most frequent to least. NOTE: That number essentially represents the absolute support for 1- itemsets! So, include a column that shows the relative support (i.e. the fraction of total transactions.) (HINT: The item with the highest support is 33.5%!)

In [40]:

```
abs = df_enc.sum().sort_values(ascending=False)
rel = abs / df_enc.count()[0] * 100

df_supp = pd.DataFrame({'count': abs,'relative count':rel})
df_supp
```

Out[40]:

	count	relative count
Chicken Bowl	615	33.533261
Chicken Burrito	489	26.663032
Chips and Guacamole	474	25.845147
Steak Burrito	342	18.647764
Canned Soft Drink	276	15.049073
Chips	208	11.341330
Steak Bowl	188	10.250818
Bottled Water	154	8.396947
Chips and Fresh Tomato Salsa	110	5.997819
Chicken Soft Tacos	107	5.834242
Side of Chips	101	5.507088
Chicken Salad Bowl	98	5.343511
Canned Soda	94	5.125409
Veggie Burrito	91	4.961832
Barbacoa Burrito	88	4.798255
Veggie Bowl	82	4.471101
Carnitas Bowl	67	3.653217
Barbacoa Bowl	61	3.326063
Carnitas Burrito	58	3.162486
Steak Soft Tacos	54	2.944384
6 Pack Soft Drink	54	2.944384
Chips and Tomatillo Red Chili Salsa	46	2.508179
Chicken Crispy Tacos	45	2.453653
Chips and Tomatillo Green Chili Salsa	43	2.344602
Carnitas Soft Tacos	38	2.071974
Steak Crispy Tacos	35	1.908397
Chips and Tomatillo-Green Chili Salsa	31	1.690294
Steak Salad Bowl	28	1.526718
Nantucket Nectar	26	1.417666
Barbacoa Soft Tacos	25	1.363141
Chips and Roasted Chili Corn Salsa	22	1.199564
Chips and Tomatillo-Red Chili Salsa	19	1.035987
Veggie Salad Bowl	18	0.981461
Chips and Roasted Chili-Corn Salsa	18	0.981461

	count	relative count
Izze	17	0.926936
Barbacoa Crispy Tacos	11	0.599782
Barbacoa Salad Bowl	9	0.490731
Chicken Salad	9	0.490731
Carnitas Crispy Tacos	7	0.381679
Veggie Soft Tacos	7	0.381679
Veggie Salad	6	0.327154
Carnitas Salad Bowl	6	0.327154
Burrito	4	0.218103
Steak Salad	4	0.218103
Bowl	2	0.109051
Salad	1	0.054526
Crispy Tacos	1	0.054526
Chips and Mild Fresh Tomato Salsa	1	0.054526
Carnitas Salad	1	0.054526
Veggie Crispy Tacos	1	0.054526

39) Now, make a smarter decision. Like many large, real-world transaction datasets, data is sparse! You have many variables, and most observations use only a handful of them. This is the definition of a sparse dataset. You need a better minsup value. Regenerate frequent itemsets, but now use a minsup of 0.005. How many frequent itemsets were reported? Report your frequent items sorted by decreasing support order.

In [41]:
```
df_apri = apriori(df_enc, min_support=0.005, use_colnames=True)
display(str(len(df_apri)) + " frequent itemsets")
df_apri.sort_values(by="support", ascending=False)
```

```
'146 frequent itemsets'
```

Out[41]:

	support	itemsets
11	0.335333	(Chicken Bowl)
12	0.266630	(Chicken Burrito)
18	0.258451	(Chips and Guacamole)
29	0.186478	(Steak Burrito)
7	0.150491	(Canned Soft Drink)
...	...	...
109	0.005453	(Chips and Guacamole, Chicken Crispy Tacos)
132	0.005453	(Steak Soft Tacos, Steak Burrito)
131	0.005453	(Steak Bowl, Veggie Bowl)
112	0.005453	(Steak Bowl, Chicken Salad Bowl)
119	0.005453	(Chips and Guacamole, Chips and Fresh Tomato S...

146 rows × 2 columns

40) In the context of association rules, explain the difference between support, confidence, lift, leverage and conviction.

Support
Support is a measure of frequency of a item within the database.

Confidence
This shows the relationship of the antecedent and consequence where a 1 value means they always show up together, but while still holding the directionality.

Lift
List is used to see independence between the antecedent and consequent. When it is 1, then we can say that the two are independent.

Leverage
Leverage is also used to see levels of independence, if it is 0, then it shows independence.

Conviction The consequent's dependence on the antecedent is correlated with high levels of conviction. Thus if conviction is 1 then it is independent.

41) Generate all association rules that meet a minimum support of 0.01. How many rules were output in total?

In [42]:
```
rules = association_rules(df_apri, metric="confidence", min_threshold=0.
01)
str(len(rules)) + " rules"
```

Out[42]: '272 rules'

42) Show only the rules that have a lift > 2, but sorted in order of decreasing confidence. What is your strongest rule?

In [43]:
```
"""
The highest confidence is the
{'Canned Soft Drink', 'Chips and Tomatillo Red Chili Salsa'}
->
{'Chicken Bowl'}

with the score of 0.83
"""

sorted_rules = rules[rules.lift > 2].sort_values(by="confidence", ascend
ing=False)
sorted_rules.head()
```

Out[43]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverag
218	(Canned Soft Drink, Chips and Tomatillo Red Ch...	(Chicken Bowl)	0.009815	0.335333	0.008179	0.833333	2.485095	0.0048
194	(Bottled Water, Chips)	(Chicken Bowl)	0.019084	0.335333	0.015267	0.800000	2.385691	0.0088
254	(Chips and Guacamole, Chips)	(Chicken Bowl)	0.009815	0.335333	0.007088	0.722222	2.153749	0.0037
219	(Chips and Tomatillo Red Chili Salsa, Chicken ...	(Canned Soft Drink)	0.015812	0.150491	0.008179	0.517241	3.437031	0.0057
195	(Bottled Water, Chicken Bowl)	(Chips)	0.037623	0.113413	0.015267	0.405797	3.578038	0.0110

43) Consider yourself the data scientist hired to help Chipotle understand item purchasing patterns. Interpret the following rule for the non data scientist. Be careful not to say, "if your customers purchase bottled water and chips, they are also going to buy chicken bowls." Think! What do strong association rules convey?

We can say that there is an 80% probability of seeing a chicken bowl in a transaction where it also had ordered bottled water and chips.
The lift score is not too close to 1, which means there is a form of dependance between the two groups of items.

44) Suppose your boss is interested in what items are most likely related to a purchase of "Chips and Guacamole". Using your rule set generated, first select the rules that have "Chips and Guacamole" listed in the consequent itemset. Sort the rules by confidence, then by lift. And interpret your findings. Identify the item(s) that are the most suggestive of including "Chips and Guacamole" when purchased, and state why.

In [44]:
```
"""
Firstly, almost all of the high confidence ones only had
 chips and guac as the only item as the consequent item.
 This means that it is ordered individually with a main
 item.
 Here, the top five sorted antecedents can be seen in two
 ways, one is big spenders, and the other is people who
 order veggie. Big spenders can be seen in the first two,
 (236, 260, 4). The veggie burrito and bowl (182,1800
 mean that they want guac as a side to supplement the
 lightness of just  veggies.
"""
guac_rules = rules[rules.consequents.apply(lambda x: "Chips and Guacamol
e" in x)].sort_values(by=["confidence", "lift"], ascending=False)
guac_rules.head()
```

Out[44]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverag
238	(Chicken Soft Tacos, Canned Soft Drink)	(Chips and Guacamole)	0.011450	0.258451	0.005453	0.476190	1.842475	0.0024
262	(Steak Bowl, Chicken Bowl)	(Chips and Guacamole)	0.015812	0.258451	0.006543	0.413793	1.601048	0.0024
5	(6 Pack Soft Drink)	(Chips and Guacamole)	0.029444	0.258451	0.011450	0.388889	1.504688	0.0038
183	(Veggie Burrito)	(Chips and Guacamole)	0.049618	0.258451	0.016903	0.340659	1.318079	0.0040
181	(Veggie Bowl)	(Chips and Guacamole)	0.044711	0.258451	0.014722	0.329268	1.274004	0.0031

45) Process all of the items listed in the "choice_description" field by creating a new transaction dataset representing binary encoded transaction data for only Chicken Bowls.

In [45]:
```
# function to use manipulate strings to flatten with three passes
def flatten(list_string):
    output = list_string.replace("]", "").replace("[", "")
    return output.split(",")
```

In [46]:
```
# get all chicken bowls and duplicate rows with quantity over 1, then fl
atten to list of lists
chick_bowls = df_chip[df_chip.item_name == "Chicken Bowl"]
chick_bowls = chick_bowls.choice_description.repeat(chick_bowls.quantity
)
chick_bowls = chick_bowls.apply(flatten)
chick_bowls.head()
```

Out[46]:
```
4     [Tomatillo-Red Chili Salsa (Hot),  Black Beans...
4     [Tomatillo-Red Chili Salsa (Hot),  Black Beans...
5     [Fresh Tomato Salsa (Mild),  Rice,  Cheese,  S...
13    [Fresh Tomato Salsa,  Fajita Vegetables,  Rice...
19    [Tomatillo Red Chili Salsa,  Fajita Vegetables...
Name: choice_description, dtype: object
```

In [47]:
```
# binary encoding
te_ary = te.fit_transform(chick_bowls)
chick_bowls_enc = pd.DataFrame(te_ary, columns=te.columns_)
chick_bowls_enc.head()
```

Out[47]:

	Black Beans	Cheese	Fajita Vegetables	Fajita Veggies	Fresh Tomato Salsa (Mild)	Guacamole	Lettuce	Pinto Beans	Rice	Roasted Chili Corn Salsa (Medium)
0	True	True	False	False	False	False	False	False	True	False
1	True	True	False	False	False	False	False	False	True	False
2	False	True	False	False	False	True	True	False	True	False
3	False	True	True	False	False	True	False	False	True	False
4	True	True	True	False	False	False	True	False	False	False

5 rows × 23 columns

46) Use your own knowledge to generate strong frequent patterns and association rules for the choice_description items used with Chicken Bowls. Explain your findings.

In [48]:
```
# unclean data 'rice vs 'white rice'
abs = chick_bowls_enc.sum().sort_values(ascending=False)
rel = abs / chick_bowls_enc.count()[0] * 100
chick_bowls_supp = pd.DataFrame({'count': abs,'relative count':rel})
chick_bowls_supp
```

Out[48]:

	count	relative count
Rice	717	94.218134
Cheese	570	74.901445
Lettuce	456	59.921156
Sour Cream	444	58.344284
Black Beans	398	52.299606
Fresh Tomato Salsa	338	44.415243
Guacamole	294	38.633377
Fajita Vegetables	215	28.252300
Pinto Beans	128	16.819974
Roasted Chili Corn Salsa	107	14.060447
Fajita Veggies	105	13.797635
Fresh Tomato Salsa (Mild)	75	9.855453
Tomatillo Green Chili Salsa	65	8.541393
Tomatillo Red Chili Salsa	62	8.147175
Tomatillo-Red Chili Salsa (Hot)	48	6.307490
Roasted Chili Corn Salsa (Medium)	46	6.044678
Roasted Chili Corn Salsa (Medium)	28	3.679369
Tomatillo-Red Chili Salsa (Hot)	25	3.285151
Fresh Tomato Salsa (Mild)	18	2.365309
Tomatillo-Green Chili Salsa (Medium)	14	1.839685
Tomatillo-Green Chili Salsa (Medium)	10	1.314060
Fresh Tomato (Mild)	4	0.525624
White Rice	2	0.262812

In [51]:

```
"""
Rice and Cheese are the most popular and staple items
 in chicken bowls.
"""
min_supp = 0.4
min_thresh = 0.8
lift_thresh = 1


chick_bowls_apri = apriori(chick_bowls_enc, min_support=min_supp, use_co
lnames=True)
display(chick_bowls_apri.sort_values(by="support", ascending=False).head
())

chick_bowls_rules = association_rules(chick_bowls_apri, metric="confiden
ce", min_threshold=min_thresh)
chick_bowls_rules[chick_bowls_rules.lift > lift_thresh].sort_values(by=
"confidence", ascending=False)
```

	support	itemsets
3	0.942181	(Rice)
1	0.749014	(Cheese)
9	0.718791	(Rice, Cheese)
2	0.599212	(Lettuce)
4	0.583443	(Sour Cream)

Out[51]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
12	(Sour Cream, Cheese)	(Rice)	0.503285	0.942181	0.492773	0.979112	1.039197	0.018587
10	(Lettuce, Cheese)	(Rice)	0.475690	0.942181	0.465177	0.977901	1.037911	0.016991
5	(Sour Cream)	(Rice)	0.583443	0.942181	0.567674	0.972973	1.032681	0.017965
8	(Cheese, Black Beans)	(Rice)	0.431012	0.942181	0.416557	0.966463	1.025772	0.010466
4	(Lettuce)	(Rice)	0.599212	0.942181	0.578187	0.964912	1.024126	0.013621
2	(Cheese)	(Rice)	0.749014	0.942181	0.718791	0.959649	1.018540	0.013084
1	(Black Beans)	(Rice)	0.522996	0.942181	0.499343	0.954774	1.013365	0.006586
6	(Fresh Tomato Salsa)	(Rice)	0.444152	0.942181	0.423127	0.952663	1.011125	0.004655
11	(Rice, Sour Cream)	(Cheese)	0.567674	0.749014	0.492773	0.868056	1.158930	0.067577
3	(Sour Cream)	(Cheese)	0.583443	0.749014	0.503285	0.862613	1.151664	0.066278
13	(Sour Cream)	(Rice, Cheese)	0.583443	0.718791	0.492773	0.844595	1.175021	0.073399
7	(Rice, Black Beans)	(Cheese)	0.499343	0.749014	0.416557	0.834211	1.113744	0.042542
0	(Black Beans)	(Cheese)	0.522996	0.749014	0.431012	0.824121	1.100273	0.039280
9	(Rice, Lettuce)	(Cheese)	0.578187	0.749014	0.465177	0.804545	1.074139	0.032107