DMPM Lab 12

Saniya Inamdar

SRN: 201900913

Roll no.: 17

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv("D:/TY sem6/DMPM LAB/lab 12/MBO.csv", header=None)
```

In [3]:
```
df.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antiox
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

In [4]:
```
df.shape
```

Out[4]:
```
(7501, 20)
```

In [5]:
```python
transactions = df.values.reshape(-1).tolist()
```

In [6]:
```python
transactions
```

Out[6]:
```
['shrimp',
 'almonds',
 'avocado',
 'vegetables mix',
 'green grapes',
 'whole weat flour',
 'yams',
 'cottage cheese',
 'energy drink',
 'tomato juice',
 'low fat yogurt',
 'green tea',
 'honey',
 'salad',
 'mineral water',
 'salmon',
 'antioxydant juice',
 'frozen smoothie',
 'spinach',
```

In [7]:
```python
df2 = pd.DataFrame(transactions)
df2['Count']=1
df2.head()
```

Out[7]:

	0	Count
0	shrimp	1
1	almonds	1
2	avocado	1
3	vegetables mix	1
4	green grapes	1

In [8]:
```
df2=df2.groupby(by=[0], as_index=False).count().sort_values(by=['Count'], ascending=True) # count
df2['Percentage'] = (df2['Count'] / df2['Count'].sum()) # percentage
df2=df2.rename(columns={0 : 'Item'})
```

In [9]:
```
df2 = df2.reset_index().drop('index',axis=1)
```

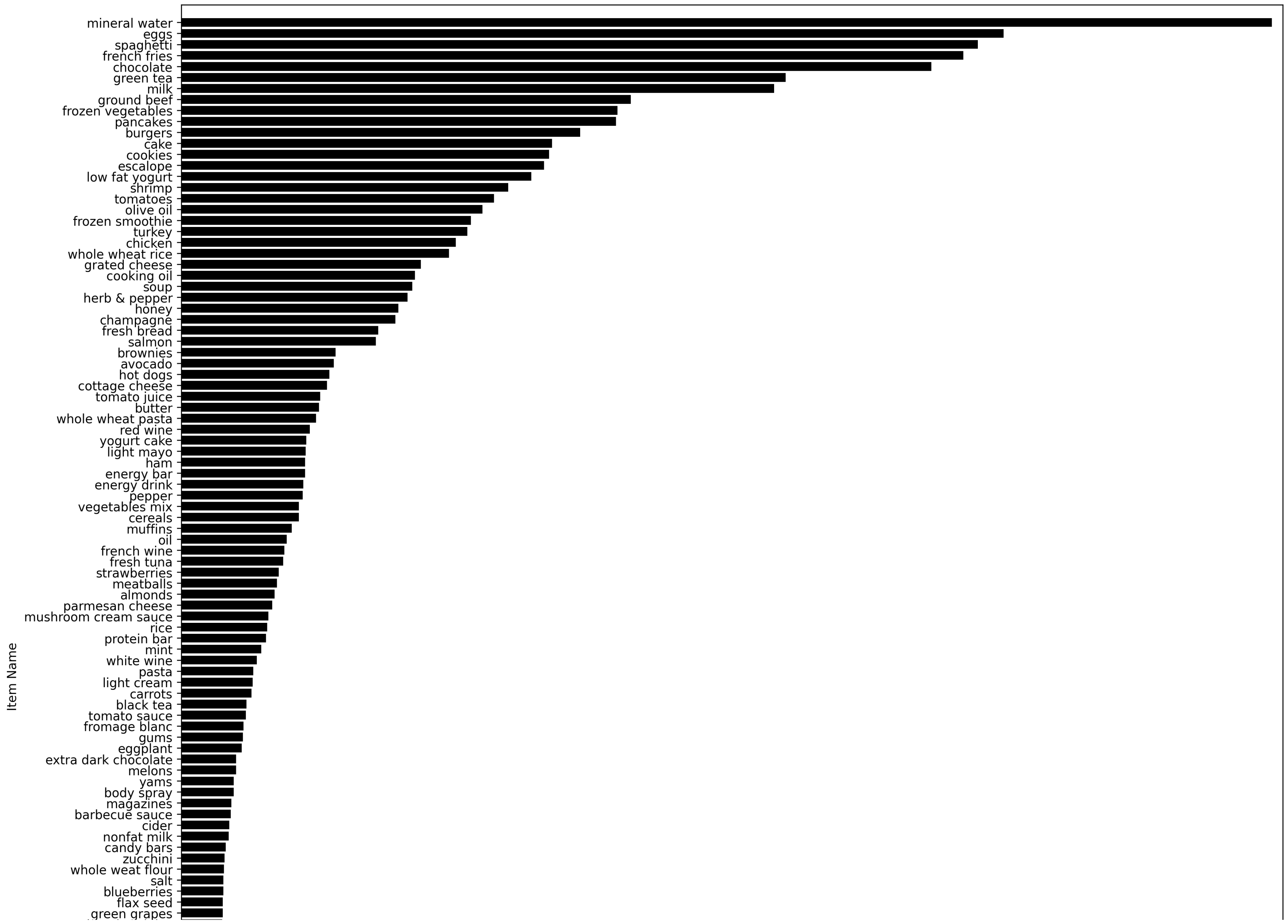
In [10]:
```
df2
```

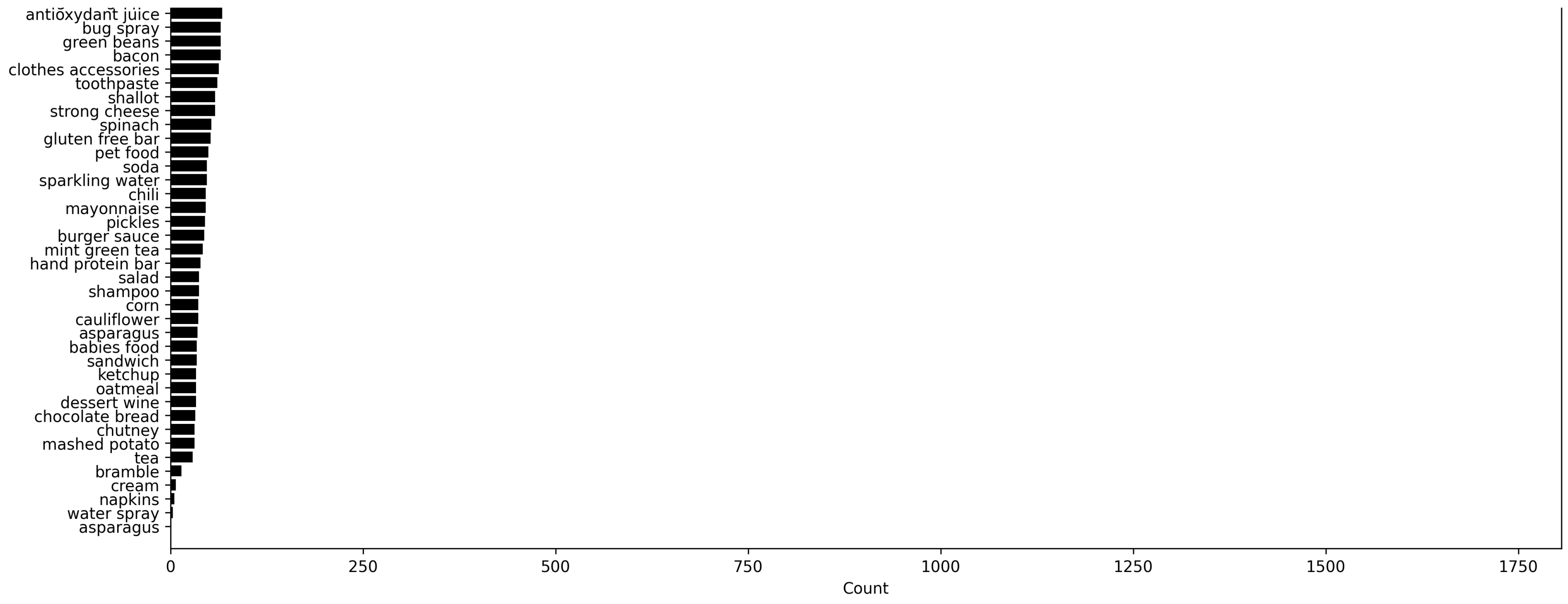
Out[10]:

	Item	Count	Percentage
0	asparagus	1	0.000034
1	water spray	3	0.000102
2	napkins	5	0.000170
3	cream	7	0.000238
4	bramble	14	0.000477
...	...	...	...
115	chocolate	1230	0.041889
116	french fries	1282	0.043660
117	spaghetti	1306	0.044478
118	eggs	1348	0.045908
119	mineral water	1788	0.060893

120 rows × 3 columns

In [11]:
```python
plt.figure(figsize=(16,20), dpi=300)
plt.ylabel('Item Name')
plt.xlabel('Count')
plt.barh(df2['Item'], width=df2['Count'], color='black', height=0.8)
plt.margins(0.01)
plt.show()
```





In [12]:
```
from efficient_apriori import apriori
```

```
In [13]:

txns2=df.stack().groupby(level=0).apply(list).tolist()

txns2
```

```
Out[13]: [['shrimp',
           'almonds',
           'avocado',
           'vegetables mix',
           'green grapes',
           'whole weat flour',
           'yams',
           'cottage cheese',
           'energy drink',
           'tomato juice',
           'low fat yogurt',
           'green tea',
           'honey',
           'salad',
           'mineral water',
           'salmon',
           'antioxydant juice',
           'frozen smoothie',
           'spinach',
```

s= 1, confidence = 30

In [14]:
```
itemsets, rules = apriori(txns2, min_support=0.01, min_confidence=0.3, verbosity=1)
```

```
Generating itemsets.
 Counting itemsets of length 1.
  Found 120 candidate itemsets of length 1.
  Found 75 large itemsets of length 1.
 Counting itemsets of length 2.
  Found 2775 candidate itemsets of length 2.
  Found 165 large itemsets of length 2.
 Counting itemsets of length 3.
  Found 477 candidate itemsets of length 3.
  Found 17 large itemsets of length 3.
 Counting itemsets of length 4.
  Found 2 candidate itemsets of length 4.
Itemset generation terminated.

Generating rules from itemsets.
 Generating rules of size 2.
 Generating rules of size 3.
Rule generation terminated.
```

In [15]:
```
for item in sorted(rules, key=lambda item: (item.lift), reverse=True)[:5]:
    print(item)

len(rules)
```

```
{herb & pepper} -> {ground beef} (conf: 0.323, supp: 0.016, lift: 3.292, conv: 1.333)
{ground beef, mineral water} -> {spaghetti} (conf: 0.417, supp: 0.017, lift: 2.395, conv: 1.416)
{frozen vegetables, mineral water} -> {milk} (conf: 0.310, supp: 0.011, lift: 2.390, conv: 1.261)
{soup} -> {milk} (conf: 0.301, supp: 0.015, lift: 2.321, conv: 1.245)
{ground beef} -> {spaghetti} (conf: 0.399, supp: 0.039, lift: 2.291, conv: 1.374)
```

Out[15]:
```
63
```

s=2% , confidence = 40%

```
In [16]: itemsets2, rules2 = apriori(txns2, min_support=0.02, min_confidence=0.4, verbosity=1)
```

```
Generating itemsets.
 Counting itemsets of length 1.
  Found 120 candidate itemsets of length 1.
  Found 53 large itemsets of length 1.
 Counting itemsets of length 2.
  Found 1378 candidate itemsets of length 2.
  Found 50 large itemsets of length 2.
 Counting itemsets of length 3.
  Found 61 candidate itemsets of length 3.
Itemset generation terminated.

Generating rules from itemsets.
 Generating rules of size 2.
Rule generation terminated.
```

```
In [17]: for item in sorted(rules2, key=lambda item: (item.lift), reverse=True)[:5]:
             print(item.lhs+item.rhs)
         len(rules2)
```

```
('soup', 'mineral water')
('olive oil', 'mineral water')
('ground beef', 'mineral water')
```

```
Out[17]: 3
```

S= 3%, confidence=50%

In [18]:
```
itemsets3, rules3 = apriori(txns2, min_support=0.03, min_confidence=0.5, verbosity=1)
```

```
Generating itemsets.
 Counting itemsets of length 1.
  Found 120 candidate itemsets of length 1.
  Found 36 large itemsets of length 1.
 Counting itemsets of length 2.
  Found 630 candidate itemsets of length 2.
  Found 18 large itemsets of length 2.
 Counting itemsets of length 3.
  Found 14 candidate itemsets of length 3.
Itemset generation terminated.

Generating rules from itemsets.
 Generating rules of size 2.
Rule generation terminated.
```

In [19]:
```
for item in sorted(rules3, key=lambda item: (item.lift), reverse=True)[:5]:
    print(item)
len(rules3)
```

Out[19]:
```
0
```

In [20]:
```
rules[1]
```

Out[20]:
```
{burgers} -> {eggs}
```

In [21]:
```
dict = {'Rules':[],
        'Item1':[],
        'Item2':[],
        'Confidence':[],
        'Support':[],
        "Lift":[]
       }

data = pd.DataFrame(dict)

for item in sorted(rules, key=lambda item: (item.lift,item.conviction), reverse=True):
    data.loc[len(data.index)] = [(item.lhs+item.rhs), item.lhs,item.rhs,item.confidence, item.support,item.lift]
```

```
C:\Users\saniy\Anaconda3\lib\site-packages\numpy\core\_asarray.py:83: VisibleDeprecationWarning: Creating an nda
rray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different length
s or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```

In [22]:
```
data.shape
```

Out[22]:
```
(63, 6)
```

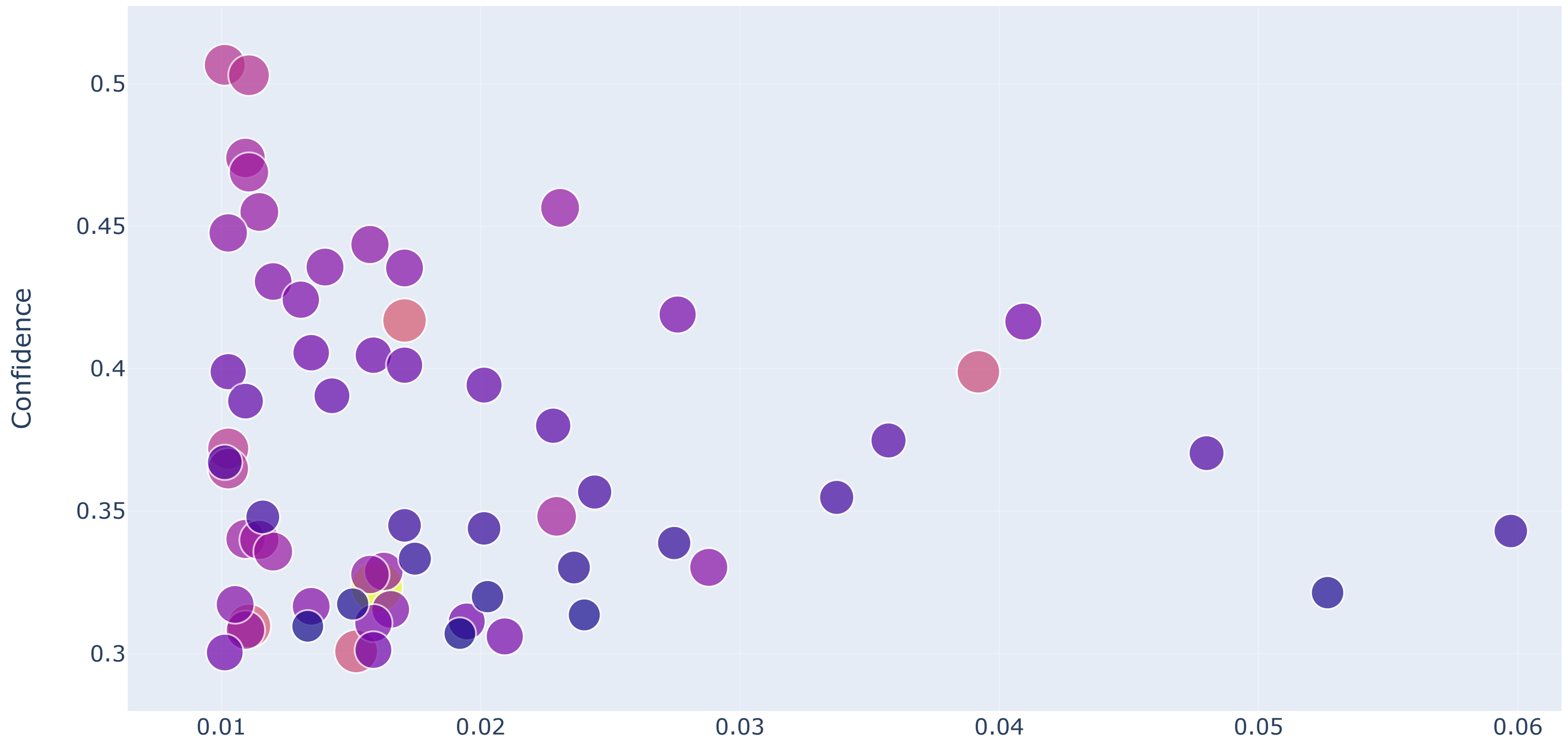
In [23]:
```
data.head()
```

Out[23]:

	Rules	Item1	Item2	Confidence	Support	Lift
0	(herb & pepper, ground beef)	(herb & pepper,)	(ground beef,)	0.323450	0.015998	3.291994
1	(ground beef, mineral water, spaghetti)	(ground beef, mineral water)	(spaghetti,)	0.416938	0.017064	2.394681
2	(frozen vegetables, mineral water, milk)	(frozen vegetables, mineral water)	(milk,)	0.309701	0.011065	2.389991
3	(soup, milk)	(soup,)	(milk,)	0.300792	0.015198	2.321232
4	(ground beef, spaghetti)	(ground beef,)	(spaghetti,)	0.398915	0.039195	2.291162

In [24]:
```
import plotly.express as px
```

```
In [25]: fig = px.scatter(data, x="Support", y="Confidence", color="Lift",
                          size='Lift')
         fig.show()
```

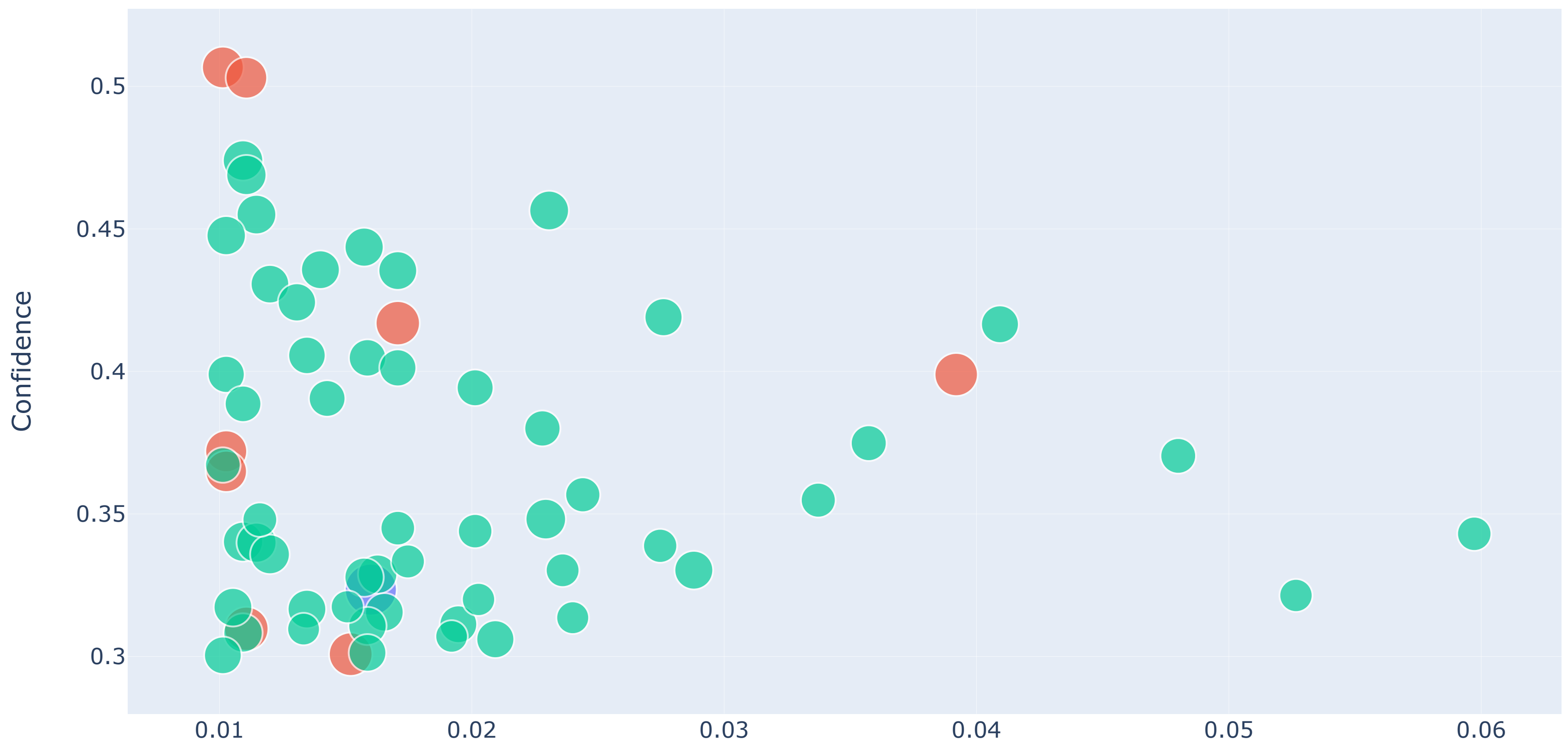


```
In [29]: data['LiftGroups'] = data['Lift'].apply(lambda x:'1-2' if 1<=x<2 else '2-3' if 2<=x<3 else 'Above 3')
In [30]: data.head()
```

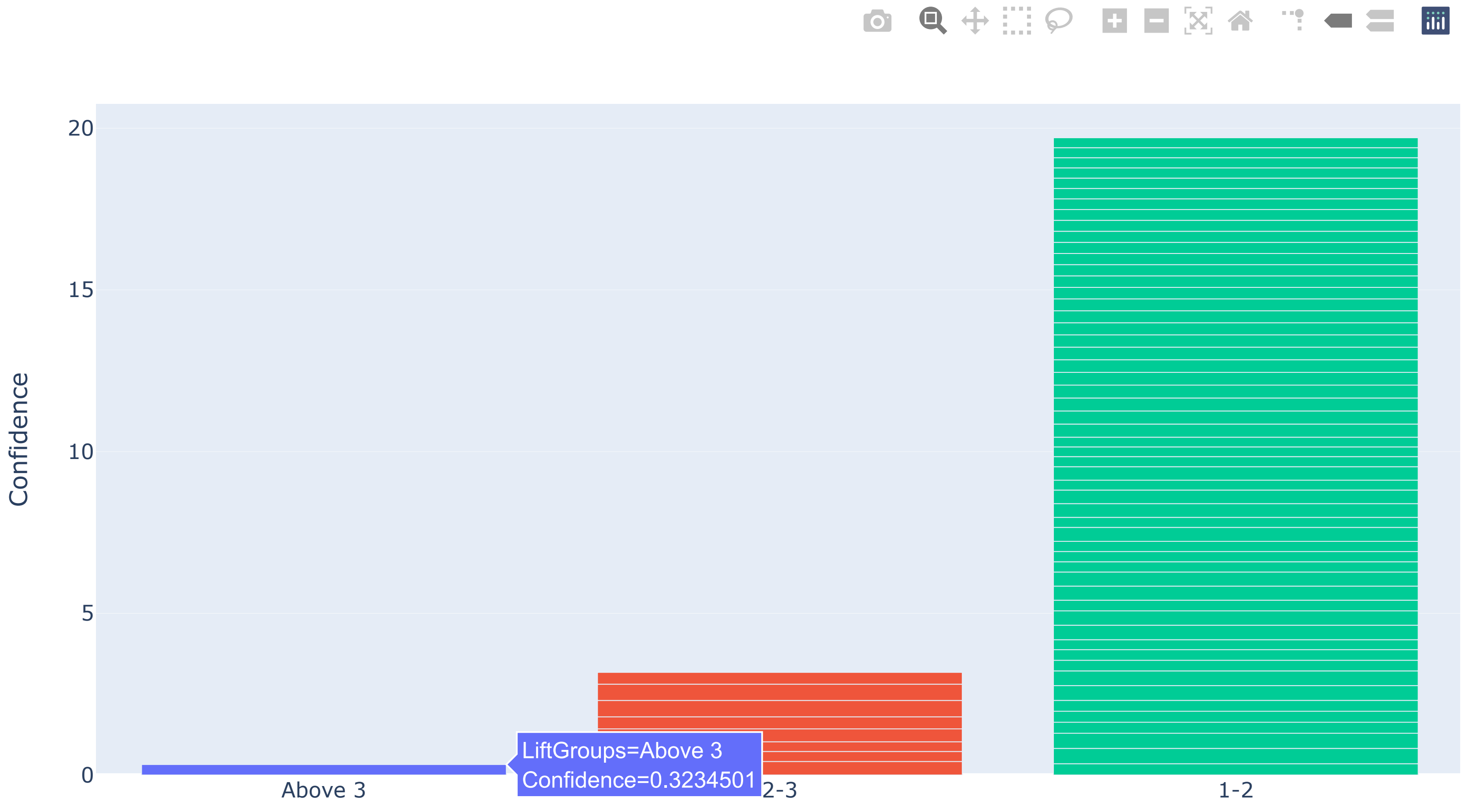
```
Out[30]:
```

	Rules	Item1	Item2	Confidence	Support	Lift	LiftGroups
0	(herb & pepper, ground beef)	(herb & pepper,)	(ground beef,)	0.323450	0.015998	3.291994	Above 3
1	(ground beef, mineral water, spaghetti)	(ground beef, mineral water)	(spaghetti,)	0.416938	0.017064	2.394681	2-3
2	(frozen vegetables, mineral water, milk)	(frozen vegetables, mineral water)	(milk,)	0.309701	0.011065	2.389991	2-3
3	(soup, milk)	(soup,)	(milk,)	0.300792	0.015198	2.321232	2-3
4	(ground beef, spaghetti)	(ground beef,)	(spaghetti,)	0.398915	0.039195	2.291162	2-3

```
In [34]: fig = px.scatter(data, x="Support", y="Confidence", color="LiftGroups",
                          size='Lift')
         fig.show()
```



```
In [36]: fig = px.bar(data, x="LiftGroups", y="Confidence", color="LiftGroups")
         fig.show()
```


Confidence
Above 3
LiftGroups=Above 3
Confidence=0.3234501