==

ASSOCIATION RULE MINING via Apriori Algorithm

In [1]:
```
#!pip install --index-url https://test.pypi.org/simple/ PyARMViz
#!pip install --upgrade networkx
```

Importing Python libraries neccessary for this case study

In [2]:
```
import pandas as pd
import numpy as np
import networkx as nx
import plotly.express as px
import matplotlib.pyplot as plt
import warnings
import seaborn as sns
from PyARMViz import PyARMViz
from networkx import convert_matrix as nxc
warnings.filterwarnings('ignore')
plt.style.use('seaborn')
```

Importing dataset

In [3]:
```
data = pd.read_csv('bread basket.csv')
```

Exploratory Data Analysis

In [4]:
```
data
```

Out[4]:

	Transaction	Item	date_time	period_day	weekday_weekend
0	1	Bread	30-10-2016 09:58	morning	weekend
1	2	Scandinavian	30-10-2016 10:05	morning	weekend
2	2	Scandinavian	30-10-2016 10:05	morning	weekend
3	3	Hot chocolate	30-10-2016 10:07	morning	weekend
4	3	Jam	30-10-2016 10:07	morning	weekend
...	...	...	...	...	...
20502	9682	Coffee	09-04-2017 14:32	afternoon	weekend
20503	9682	Tea	09-04-2017 14:32	afternoon	weekend
20504	9683	Coffee	09-04-2017 14:57	afternoon	weekend
20505	9683	Pastry	09-04-2017 14:57	afternoon	weekend
20506	9684	Smoothies	09-04-2017 15:04	afternoon	weekend

20507 rows × 5 columns

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20507 entries, 0 to 20506
Data columns (total 5 columns):
```

```
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Transaction      20507 non-null  int64
 1   Item             20507 non-null  object
 2   date_time        20507 non-null  object
 3   period_day       20507 non-null  object
 4   weekday_weekend  20507 non-null  object
dtypes: int64(1), object(4)
```

In [6]:
```python
print("Total number of unique transactions = ", len(data['Transaction'].unique()))
```

```
Total number of unique transactions =  9465
```

In [7]:
```python
print("Total number of unique items = ", len(data['Item'].unique()))
data['Item'].unique()
```

```
Total number of unique items =  94
```

Out[7]:
```
array(['Bread', 'Scandinavian', 'Hot chocolate', 'Jam', 'Cookies',
       'Muffin', 'Coffee', 'Pastry', 'Medialuna', 'Tea', 'Tartine',
       'Basket', 'Mineral water', 'Farm House', 'Fudge', 'Juice',
       "Ella's Kitchen Pouches", 'Victorian Sponge', 'Frittata',
       'Hearty & Seasonal', 'Soup', 'Pick and Mix Bowls', 'Smoothies',
       'Cake', 'Mighty Protein', 'Chicken sand', 'Coke',
       'My-5 Fruit Shoot', 'Focaccia', 'Sandwich', 'Alfajores', 'Eggs',
       'Brownie', 'Dulce de Leche', 'Honey', 'The BART', 'Granola',
       'Fairy Doors', 'Empanadas', 'Keeping It Local', 'Art Tray',
       'Bowl Nic Pitt', 'Bread Pudding', 'Adjustment', 'Truffles',
       'Chimichurri Oil', 'Bacon', 'Spread', 'Kids biscuit', 'Siblings',
       'Caramel bites', 'Jammie Dodgers', 'Tiffin', 'Olum & polenta',
       'Polenta', 'The Nomad', 'Hack the stack', 'Bakewell',
       'Lemon and coconut', 'Toast', 'Scone', 'Crepes', 'Vegan mincepie',
       'Bare Popcorn', 'Muesli', 'Crisps', 'Pintxos', 'Gingerbread syrup',
       'Panatone', 'Brioche and salami', 'Afternoon with the baker',
       'Salad', 'Chicken Stew', 'Spanish Brunch',
       'Raspberry shortbread sandwich', 'Extra Salami or Feta',
       'Duck egg', 'Baguette', "Valentine's card", 'Tshirt',
       'Vegan Feast', 'Postcard', 'Nomad bag', 'Chocolates',
       'Coffee granules ', 'Drinking chocolate spoons ',
       'Christmas common', 'Argentina Night', 'Half slice Monster ',
       'Gift voucher', 'Cherry me Dried fruit', 'Mortimer', 'Raw bars',
       'Tacos/Fajita'], dtype=object)
```

In [8]:
```python
print("Value counts of each items")
data['Item'].value_counts()
```

```
Value counts of each items
```

Out[8]:
```
Coffee            5471
Bread             3325
Tea               1435
Cake              1025
Pastry             856
                  ...
Bacon                1
Gift voucher         1
Olum & polenta       1
Raw bars             1
Polenta              1
Name: Item, Length: 94, dtype: int64
```

In [9]:
```python
print("Value counts of each period_day")
data['period_day'].value_counts()
```

```
Value counts of each period_day
```

Out[9]:
```
afternoon    11569
morning       8404
evening        520
night           14
Name: period_day, dtype: int64
```

In [10]:
```python
print("Value counts of weekday_weekend")
data['weekday_weekend'].value_counts()
```

```
Value counts of weekday_weekend
```

Out[10]:
```
weekday    12807
weekend     7700
```

In [11]:
```
vc = data['Item'].value_counts().reset_index()
vc = list(vc['index'][:10]) ##top 10 items
```

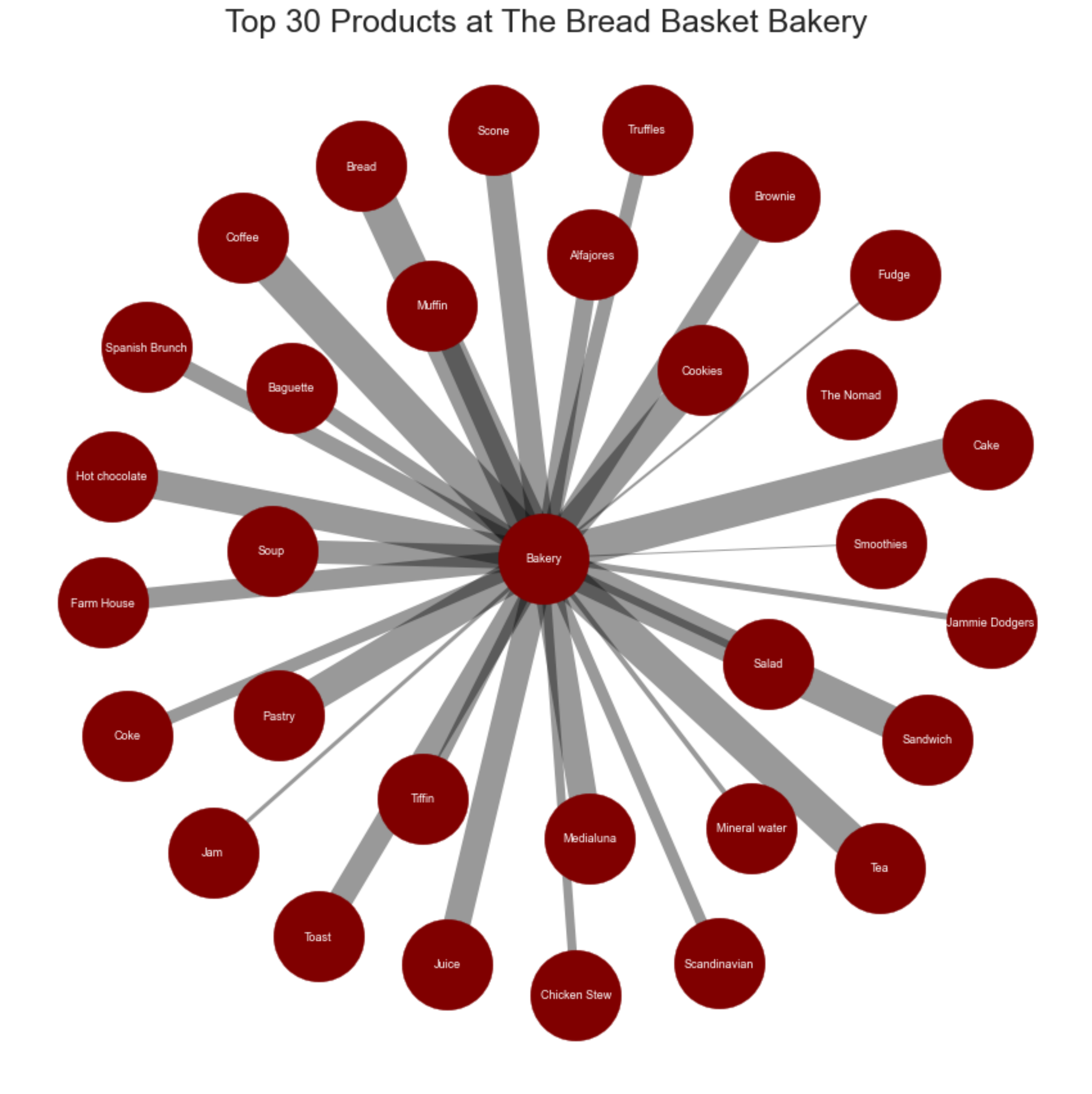
In [12]:
```
hist_data = data[['Item', 'period_day', 'weekday_weekend']]
hist_data = hist_data[hist_data.Item.isin(vc)]
fig, ax = plt.subplots(3,1,figsize=[20,15])
sns.countplot(hist_data['Item'], ax=ax[0])
sns.countplot(hist_data['period_day'], ax=ax[1])
sns.countplot(hist_data['weekday_weekend'], ax=ax[2])
```

Out[12]:
```
<AxesSubplot:xlabel='weekday_weekend', ylabel='count'>
```

We would only need Transaction, Item and date time. Let's try and explore which are the top 30 selling products in the bakery using a network diagram.

In [13]:

```
data_vis = data.copy()
df_network_first = data_vis.groupby("Item").sum().sort_values("Transaction", ascending=False).reset_index()
df_network_first["Type"] = "Bakery"
df_network_first = df_network_first.truncate(before=-1, after=30) # top 30
plt.rcParams['figure.figsize']=(15,15)
j = 0
for i, _ in reversed(list(enumerate(df_network_first['Transaction']))):
    df_network_first['Transaction'][j] = i
    j+=1
first_choice = nxc.from_pandas_edgelist(df_network_first, source='Type', target="Item", edge_attr='Transaction')
prior = [i['Transaction'] for i in dict(first_choice.edges).values()]
pos = nx.spring_layout(first_choice)
nx.draw_networkx_nodes(first_choice, pos, node_size=5000, node_color="maroon")
nx.draw_networkx_edges(first_choice, pos, width=prior, alpha=0.4, edge_color='black')
nx.draw_networkx_labels(first_choice, pos, font_size=9, font_family='sans-serif', font_color = 'white')
plt.axis('off')
plt.grid()
plt.title('Top 30 Products at The Bread Basket Bakery', fontsize=25)
plt.show()
```



Data Preprocessing

Its time to model the Apriori algorithm. We should first generate the frequent item set and then generate the association rules using the frequent item set. We need to ensure we generate a matrix with 0/1 values representating transaction presence of that item.

In [14]:
```
from mlxtend.frequent_patterns import association_rules, apriori

def encoder(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

apriori_data = data.groupby(['Transaction','Item'])['Item'].count().reset_index(name ='Count')
apriori_data.head(15)
```

Out[14]:

Transaction	Item	Count

	Transaction	Item	Count
0	1	Bread	1
1	2	Scandinavian	2
2	3	Cookies	1
3	3	Hot chocolate	1
4	3	Jam	1
5	4	Muffin	1
6	5	Bread	1
7	5	Coffee	1
8	5	Pastry	1
9	6	Medialuna	1
10	6	Muffin	1
11	6	Pastry	1
12	7	Coffee	1
13	7	Medialuna	1

In [15]:

```
apriori_basket = apriori_data.pivot_table(index = 'Transaction', columns = 'Item', values = 'Count', aggfunc = 'sum').fillna(0)
apriori_basket_set = apriori_basket.applymap(encoder)
apriori_basket_set
```

Out[15]:

Item	Adjustment	Afternoon with the baker	Alfajores	Argentina Night	Art Tray	Bacon	Baguette	Bakewell	Bare Popcorn	Basket	...	The BART	The Nomad	Tiffin	Toast	Truffles	Tshirt	Valentine's card	Vegan Feast	Vegan mincepie	Victorian Sponge
Transaction																					
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9680	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9681	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	0	0
9682	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9683	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9684	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

9465 rows × 94 columns

Let's first analyze the rules with min_support 5% and then for 1% respectively. Both using the metric lift.

In [16]:

```
f_items = apriori(apriori_basket_set, min_support = 0.05, use_colnames = True)
f_items
```

Out[16]:

	support	itemsets
0	0.327205	(Bread)
1	0.103856	(Cake)
2	0.478394	(Coffee)
3	0.054411	(Cookies)
4	0.058320	(Hot chocolate)

	support	itemsets
5	0.061807	(Medialuna)
6	0.086107	(Pastry)
7	0.071844	(Sandwich)
8	0.142631	(Tea)
9	0.090016	(Bread, Coffee)

In [17]:

```
apriori_rules = association_rules(f_items, metric = 'lift', min_threshold = 0.05)
apriori_rules.sort_values('confidence', ascending = False, inplace = True)
apriori_rules
```

Out[17]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(Cake)	(Coffee)	0.103856	0.478394	0.054728	0.526958	1.101515	0.005044	1.102664
0	(Bread)	(Coffee)	0.327205	0.478394	0.090016	0.275105	0.575059	-0.066517	0.719561
1	(Coffee)	(Bread)	0.478394	0.327205	0.090016	0.188163	0.575059	-0.066517	0.828731
3	(Coffee)	(Cake)	0.478394	0.103856	0.054728	0.114399	1.101515	0.005044	1.011905

In [18]:

```
f_items = apriori(apriori_basket_set, min_support = 0.01, use_colnames = True)
f_items
```

Out[18]:

	support	itemsets
0	0.036344	(Alfajores)
1	0.016059	(Baguette)
2	0.327205	(Bread)
3	0.040042	(Brownie)
4	0.103856	(Cake)
...	...	...
56	0.023666	(Toast, Coffee)
57	0.014369	(Tea, Sandwich)
58	0.010037	(Cake, Bread, Coffee)
59	0.011199	(Pastry, Bread, Coffee)
60	0.010037	(Cake, Tea, Coffee)

61 rows × 2 columns

In [19]:

```
apriori_rules = association_rules(f_items, metric = 'lift', min_threshold = 0.01)
apriori_rules.sort_values('confidence', ascending = False, inplace = True)
apriori_rules
```

Out[19]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
52	(Toast)	(Coffee)	0.033597	0.478394	0.023666	0.704403	1.472431	0.007593	1.764582
49	(Spanish Brunch)	(Coffee)	0.018172	0.478394	0.010882	0.598837	1.251766	0.002189	1.300235
37	(Medialuna)	(Coffee)	0.061807	0.478394	0.035182	0.569231	1.189878	0.005614	1.210871
40	(Pastry)	(Coffee)	0.086107	0.478394	0.047544	0.552147	1.154168	0.006351	1.164682
2	(Alfajores)	(Coffee)	0.036344	0.478394	0.019651	0.540698	1.130235	0.002264	1.135648
...	...	...	...	...	...	...	...	...	...
60	(Bread)	(Coffee, Cake)	0.327205	0.054728	0.010037	0.030675	0.560497	-0.007870	0.975186

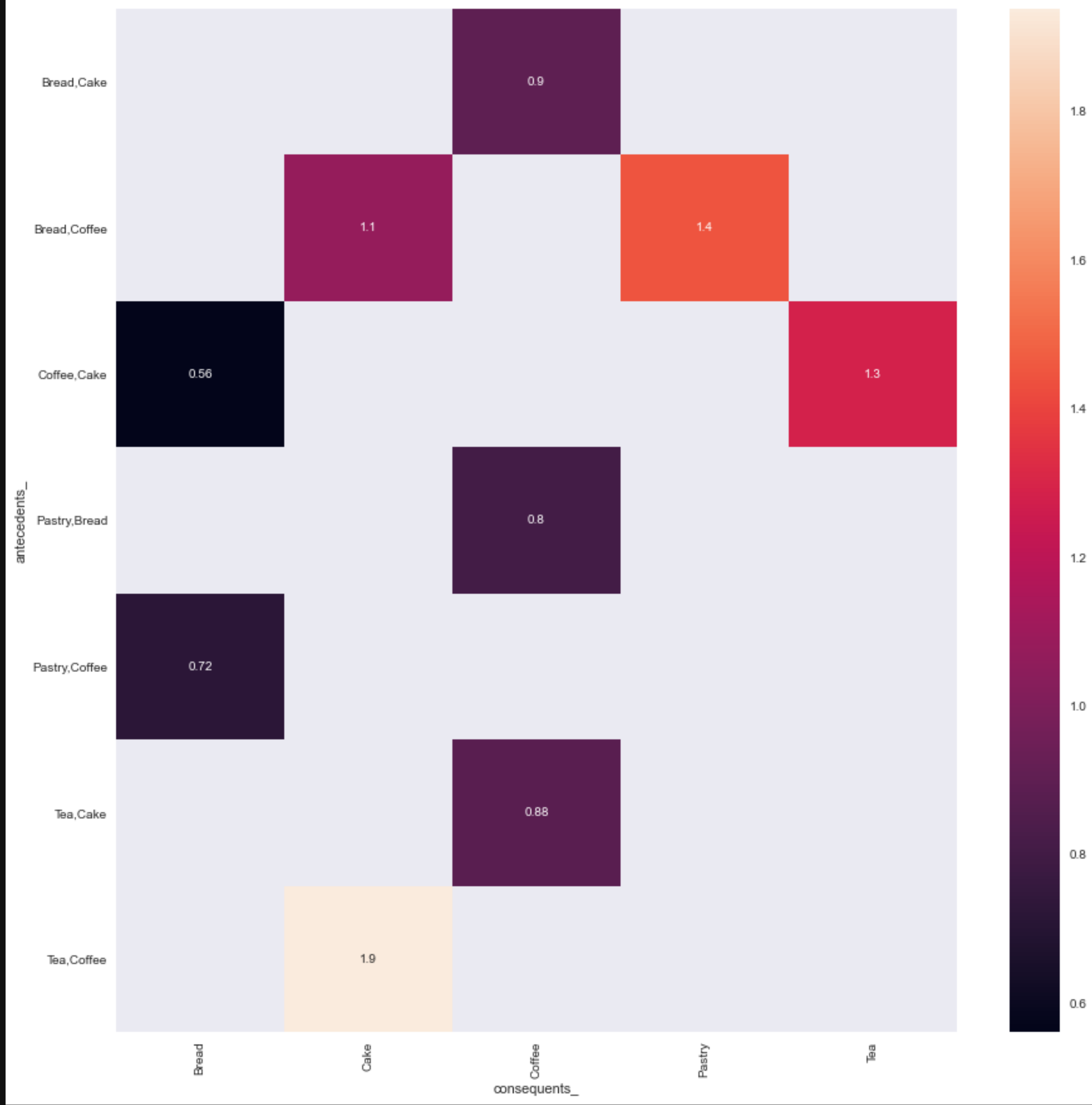
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
67	(Coffee)	(Pastry, Bread)	0.478394	0.029160	0.011199	0.023410	0.802807	-0.002751	0.994112
48	(Coffee)	(Spanish Brunch)	0.478394	0.018172	0.010882	0.022747	1.251766	0.002189	1.004682
61	(Coffee)	(Bread, Cake)	0.478394	0.023349	0.010037	0.020981	0.898557	-0.001133	0.997581
73	(Coffee)	(Tea, Cake)	0.478394	0.023772	0.010037	0.020981	0.882582	-0.001335	0.997149

Data Visualizations for Association Rules

First let's take look at a heatmap of the association rules generated for 1% minimum support threshold.

In [20]:

```
apriori_rules['lhs_items'] = apriori_rules['antecedents'].apply(lambda x:len(x) )
apriori_rules[apriori_rules['lhs_items']>1].sort_values('lift', ascending=False).head()
apriori_rules['antecedents_'] = apriori_rules['antecedents'].apply(lambda a: ','.join(list(a)))
apriori_rules['consequents_'] = apriori_rules['consequents'].apply(lambda a: ','.join(list(a)))
pivot = apriori_rules[apriori_rules['lhs_items']>1].pivot(index = 'antecedents_', columns = 'consequents_', values= 'lift')
sns.heatmap(pivot, annot = True)
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```



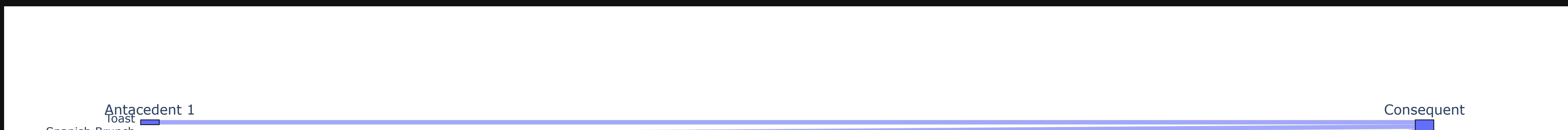
Another great visualization library I absolutely love is the PyARMViz library which can work with complex large scale rules. Let's first generate a parallel categories plot

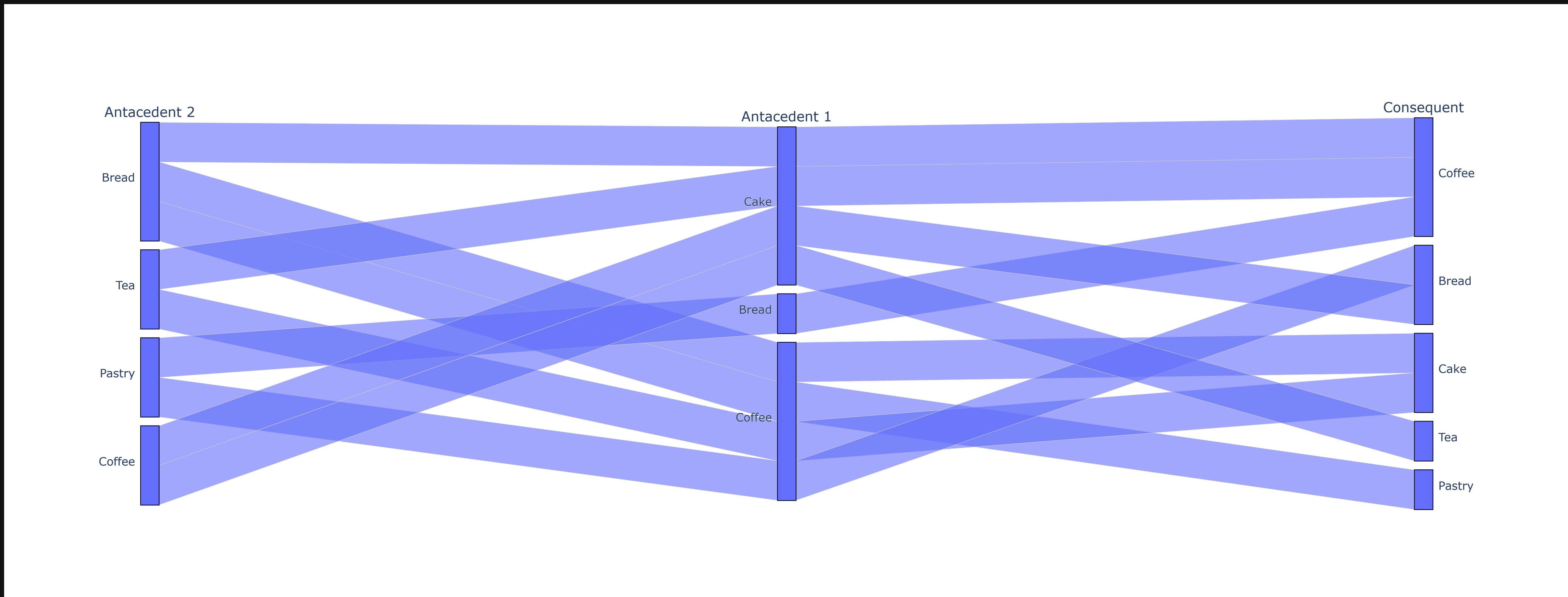
In [21]:

```python
from PyARMViz import PyARMViz
from PyARMViz.Rule import generate_rule_from_dict

apriori_vis = apriori_rules

apriori_vis['uni'] = np.nan
apriori_vis['ant'] = np.nan
apriori_vis['con'] = np.nan
apriori_vis['tot'] = 20507

transactions = [a[1]['Item'].tolist() for a in list(data.groupby(['Transaction', 'date_time']))]

def tran():
    for t in transactions:
        yield t
def antec(x):
    cnt = 0
    for t in tran():
        t = set(t)
        if x.intersection(t) == x:
            cnt = cnt + 1
    return cnt
vis = apriori_vis.values.tolist()

rules_dict = []
for i in vis:
    i[10] = antec(i[0])
    i[11] = antec(i[1])
    i[9] = antec(i[0].union(i[1]))
    diction = {
        'lhs': tuple(i[0]),
        'rhs': tuple(i[1]),
        'count_full': i[9],
        'count_lhs': i[10],
        'count_rhs': i[11],
        'num_transactions': i[12]
    }
    rules_dict.append(diction)
```

In [22]:

```python
rules = []
for rd in rules_dict:
    rules.append(generate_rule_from_dict(rd))

PyARMViz.generate_parallel_category_plot(rules)
```





Ofcourse this plot is much more useful with larger number of association rules. So let's try using the same analysis this time with 0.5% minimum support threshold

In [23]:

```
# with 0.001

f_items = apriori(apriori_basket_set, min_support = 0.005, use_colnames = True)
apriori_rules = association_rules(f_items, metric = 'lift', min_threshold = 0.005)
apriori_rules.sort_values('confidence', ascending = False, inplace = True)

apriori_vis = apriori_rules

apriori_vis['uni'] = np.nan
apriori_vis['ant'] = np.nan
apriori_vis['con'] = np.nan
apriori_vis['tot'] = 20507

transactions = [a[1]['Item'].tolist() for a in list(data.groupby(['Transaction', 'date_time']))]

def tran():
    for t in transactions:
        yield t
def antec(x):
    cnt = 0
    for t in tran():
        t = set(t)
        if x.intersection(t) == x:
            cnt = cnt + 1
    return cnt
vis = apriori_vis.values.tolist()

rules_dict = []
for i in vis:
    i[10] = antec(i[0])
    i[11] = antec(i[1])
    i[9] = antec(i[0].union(i[1]))
    diction = {
        'lhs': tuple(i[0]),
        'rhs': tuple(i[1]),
        'count_full': i[9],
        'count_lhs': i[10],
        'count_rhs': i[11],
        'num_transactions': i[12]
    }
    rules_dict.append(diction)

rules = []
for rd in rules_dict:
    rules.append(generate_rule_from_dict(rd))
```

In [24]:

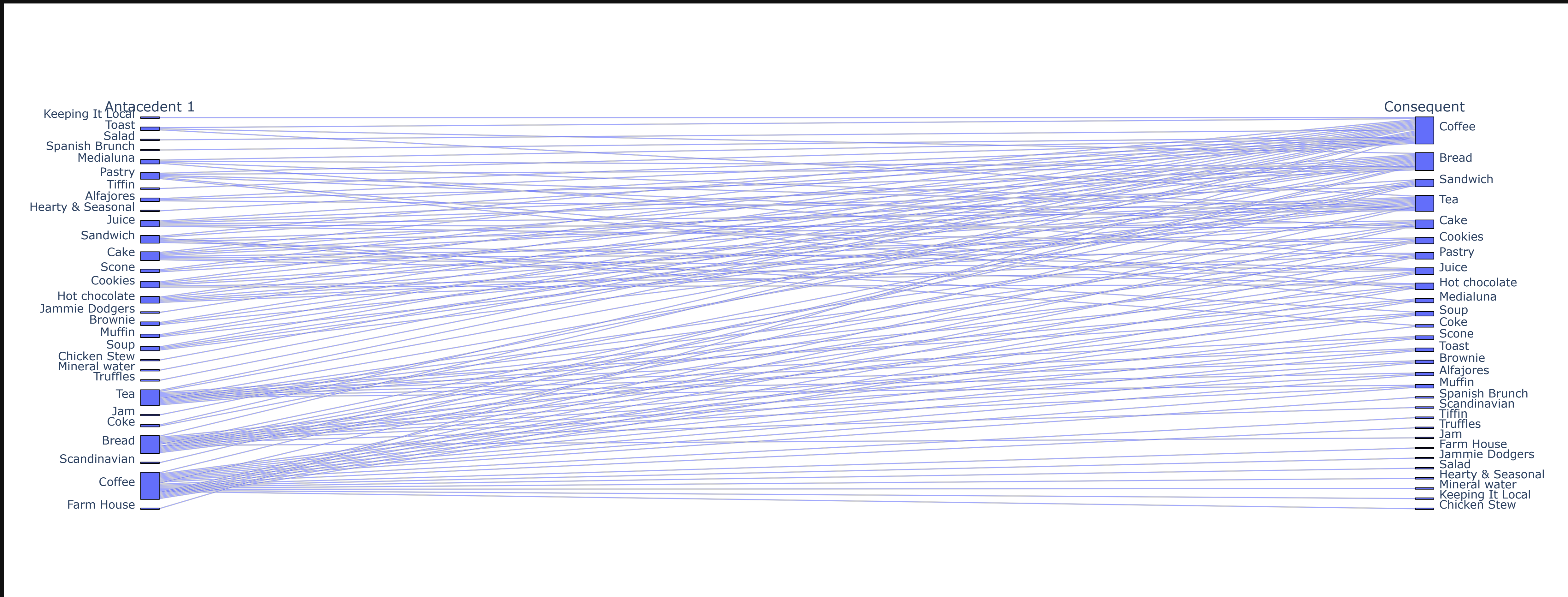
```
def enable_plotly_in_cell():
    import IPython
    from plotly.offline import init_notebook_mode
    display(IPython.core.display.HTML('''<script src="/static/components/requirejs/require.js"></script>'''))
    init_notebook_mode(connected=False)
```

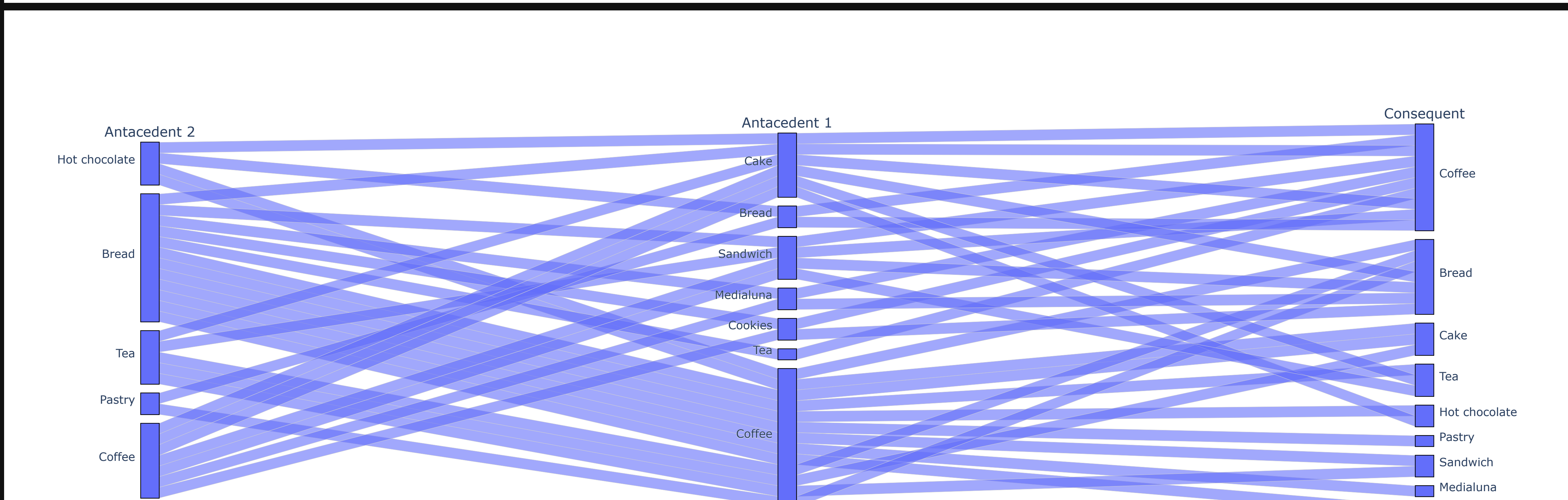
In [25]:

```
enable_plotly_in_cell()
```

In [26]:

```
PyARMViz.generate_parallel_category_plot(rules)
```

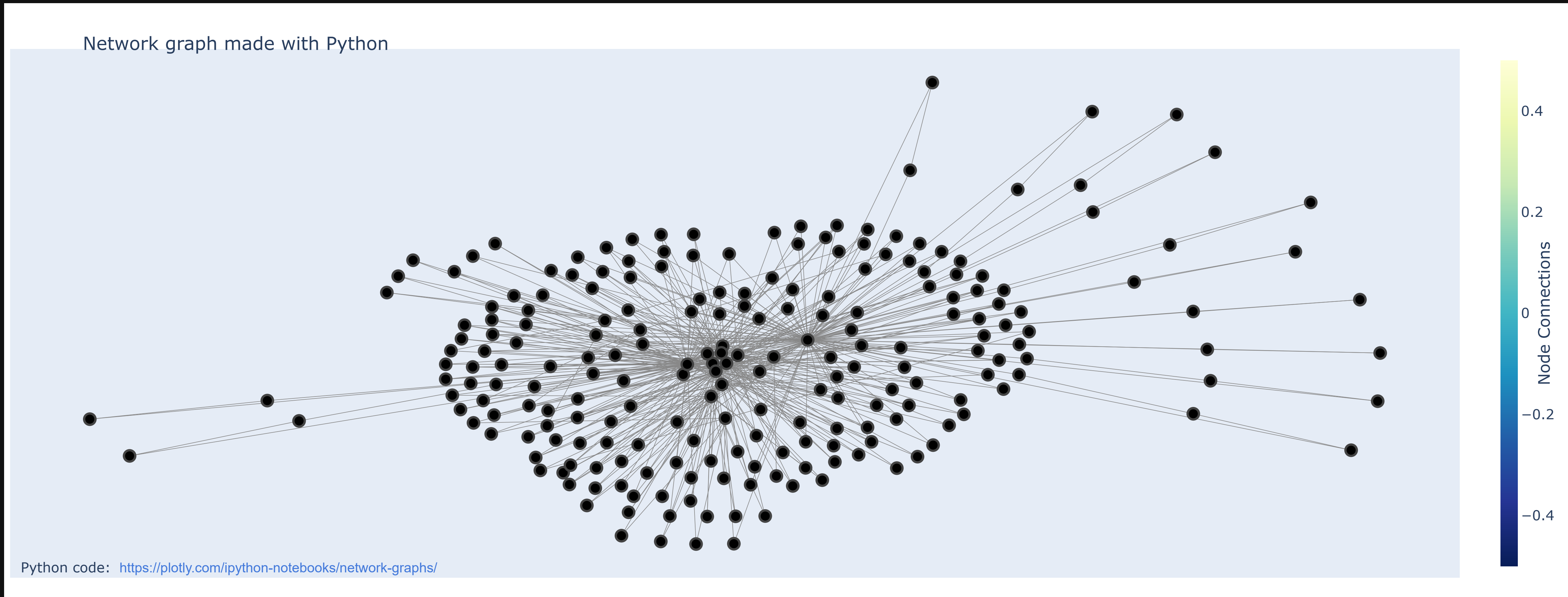




Now let's take a look at the network graph plot for the rules.enable_plotly_in_cell()

In [27]:
```
enable_plotly_in_cell()
```

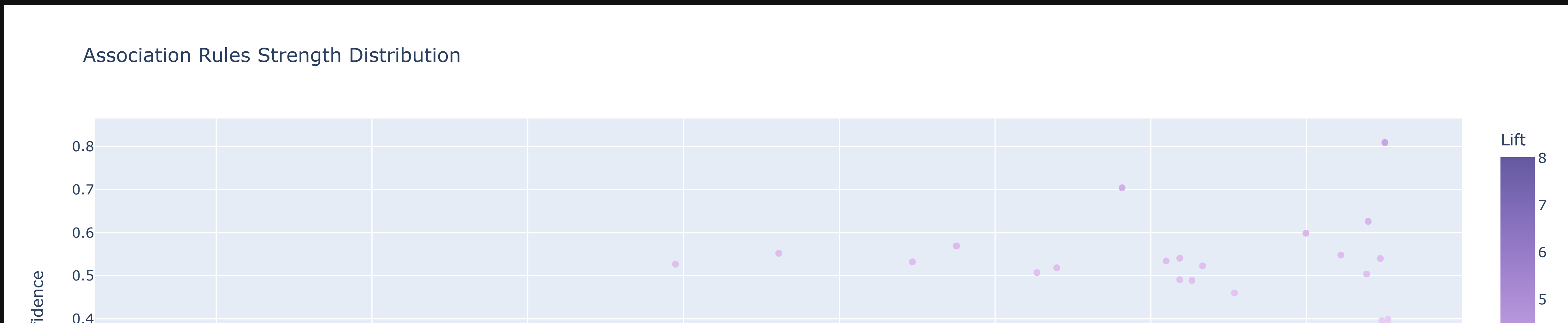
In [28]:
```
PyARMViz.generate_rule_graph_plotly(rules)
```

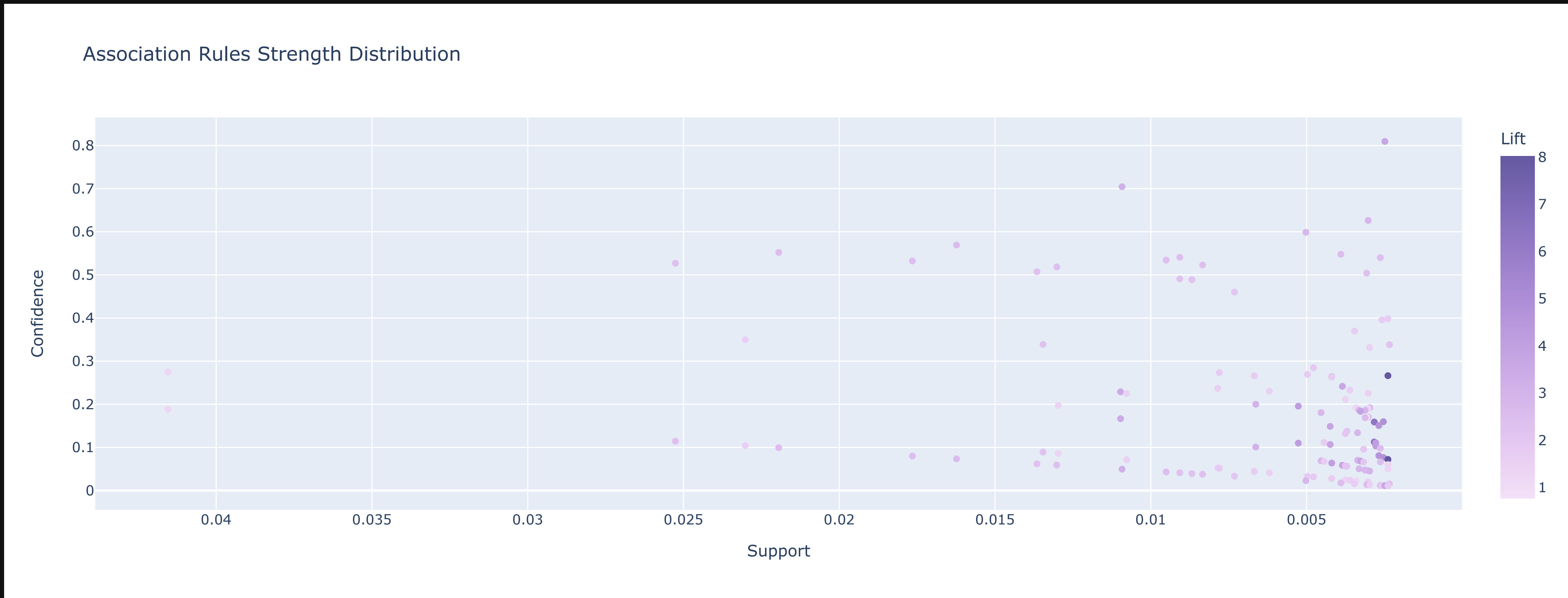


Also the association strength plot which is a scatter plot of support vs confidence with metric as the color dimension.enable_plotly_in_cell()

In [29]:
```
enable_plotly_in_cell()
```

In [30]:
```
PyARMViz.generate_rule_strength_plot(rules)
```




Association Rules Strength Distribution
Confidence
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Support
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
Lift
8
7
6
5
4
3
2
1