Associate Rule Assigment

Objective

1. To create our own dataset
2. Group Quantitative and Categorical Data
3. Find associate related to your dataset.
4. Provide your conclusion

For this assignment we will be using (apriori, association_rules) functions from mlxtend libraray.

Reading data

In [111]:

```
import numpy as np
import pandas as pd
```

In [112]:

```
df1 = pd.read_csv('Salon_Sample_Data(AR).csv')
df.head()
```

Out[112]:

	InvoiceNo	haircut	hairstylist	location	quantity
0	1	facial	hasim	achole	1
1	1	anti-dandruff	viraj	achole	1
2	2	beard	kasim	achole	1
3	2	straightning	hasim	achole	1
4	3	facial	kasim	achole	1

In [113]:

```
df2 = pd.read_csv('Salon_Sample_Data(AR) sank.csv')
df2.head()
```

Out[113]:

	InvoiceNo	haircut	hairstylist	location	quantity
0	1	hair	niel	sankeshwar-nagar	1
1	1	dye	mukesh	sankeshwar-nagar	1
2	2	eyebrow	nitin	sankeshwar-nagar	1
3	2	facial	mukesh	sankeshwar-nagar	1
4	3	hair	nitin	sankeshwar-nagar	1

In [114]:

```
df.columns
```

Out[114]:

```
Index(['InvoiceNo', 'haircut', 'hairstylist', 'location', 'quantity'], dtype='object')
```

Summary of Data

The dataframe consits of 400 entries and 5 columns

'InvoiceNo', 'haircut', 'hairstylist', 'location', 'quantity'

1. **InvoiceNo : Transaction ID**
2. **haircut : Type of haircut**
3. **hairstylist : Name of hairstylist**
4. **location : Branch location**
5. **quantity : Quantity of haircuts done**

In [115]:

```
df = pd.concat([df1, df2])
print(df.info())
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 400 entries, 0 to 199
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   InvoiceNo    400 non-null    int64 
 1   haircut      400 non-null    object
 2   hairstylist  400 non-null    object
 3   location     400 non-null    object
 4   quantity     400 non-null    int64 
dtypes: int64(2), object(3)
memory usage: 18.8+ KB
None
```

Out[115]:

	InvoiceNo	haircut	hairstylist	location	quantity
0	1	facial	hasim	achole	1
1	1	anti-dandruff	viraj	achole	1
2	2	beard	kasim	achole	1
3	2	straightning	hasim	achole	1
4	3	facial	kasim	achole	1

In [116]:

```
print(df['haircut'].value_counts())
```

```
hair             88
beard            56
dye              50
facial           48
anti-dandruff    46
eyebrow          26
curls            26
hair-spa         23
straightning     22
stylinig         15
Name: haircut, dtype: int64
```

Grouping with InvoiceNo. and Haircut-type

In [117]:

```
basket = (df[df['location']=='achole']).groupby(['InvoiceNo', 'haircut'])['quantity']\
.sum().unstack().reset_index().fillna(0)\
.set_index('InvoiceNo')
basket
```

Out[117]:

haircut	anti-dandruff	beard	curls	dye	eyebrow	facial	hair	hair-spa	straightning	stylinig
InvoiceNo										
1	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
5	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
96	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
97	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
98	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
99	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
100	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

100 rows × 10 columns

Updating the data to boolean format (1 if done else 0)

In [118]:

```
basket_sets = basket.applymap(lambda x: 1 if x>=1 else 0)
print(basket_sets)
```

```
haircut    anti-dandruff  beard  curls  ...  hair-spa  straightning  stylinig
InvoiceNo                                ...                                   
1                      1      0      0  ...         0             0         0
2                      0      1      0  ...         0             1         0
3                      1      0      0  ...         0             0         0
4                      0      0      0  ...         0             0         0
5                      0      1      0  ...         0             0         0
...                  ...    ...    ...  ...       ...           ...       ...
96                     0      1      0  ...         0             1         0
97                     0      0      0  ...         0             0         0
98                     0      0      0  ...         0             0         1
99                     1      0      0  ...         0             0         0
100                    0      0      1  ...         0             0         1

[100 rows x 10 columns]
```

Using 'apriori' and 'association_rules' to generate patters

In [119]:

```
from mlxtend.frequent_patterns import apriori, association_rules
```

Trying minimum support greater than 0 and checking for best suitable value

In this case 0.14 is a better value which also has confidence greater than 50%

In [120]:

```
frequent_itemsets = apriori(basket_sets, min_support=0.14, use_colnames=True)
print(frequent_itemsets)
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=0.8 )
rules
```

```
    support                 itemsets
0      0.18          (anti-dandruff)
1      0.30                  (beard)
```

```
2     0.27                   (dye)
3     0.17               (eyebrow)
4     0.19                (facial)
5     0.48                  (hair)
6     0.14   (facial, anti-dandruff)
7     0.17           (hair, beard)
8     0.23             (hair, dye)
```

Out[120]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(facial)	(anti-dandruff)	0.19	0.18	0.14	0.736842	4.093567	0.1058	3.116000
1	(anti-dandruff)	(facial)	0.18	0.19	0.14	0.777778	4.093567	0.1058	3.645000
2	(hair)	(beard)	0.48	0.30	0.17	0.354167	1.180556	0.0260	1.083871
3	(beard)	(hair)	0.30	0.48	0.17	0.566667	1.180556	0.0260	1.200000
4	(hair)	(dye)	0.48	0.27	0.23	0.479167	1.774691	0.1004	1.401600
5	(dye)	(hair)	0.27	0.48	0.23	0.851852	1.774691	0.1004	3.510000

Conclusion

The below data shows us most frequent choices of customers

We are using Conditional Slicing,

Where LIFT is greater than 1 and CONFIDENCE is greater than 50%.

In [127]:

```
ac_rules = rules[ (rules.lift > 1) & (rules.confidence > 0.5)]
ac_rules
```

Out[127]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(facial)	(anti-dandruff)	0.19	0.18	0.14	0.736842	4.093567	0.1058	3.116
1	(anti-dandruff)	(facial)	0.18	0.19	0.14	0.777778	4.093567	0.1058	3.645
3	(beard)	(hair)	0.30	0.48	0.17	0.566667	1.180556	0.0260	1.200
5	(dye)	(hair)	0.27	0.48	0.23	0.851852	1.774691	0.1004	3.510

In [149]:

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

labels = ['Positive', 'Negative']
choices = rules[['antecedents','consequents']]
r  = ac_rules['confidence']
fig = make_subplots(rows=1, cols=3, specs=[[{'type':'domain'}, {'type':'domain'}, {'type
':'domain'}]])
fig.add_trace(go.Pie(labels=labels, values=[r[0],1-r[0]]),
              1, 1)
fig.add_trace(go.Pie(labels=labels, values=[r[3],1-r[3]]),
              1, 2)
fig.add_trace(go.Pie(labels=labels, values=[r[5],1-r[5]]),
              1, 3)
```

```
fig.update_traces(hole=.4, hoverinfo="percent+name")

fig.update_layout(
    title_text="Associate Rules",
    annotations=[dict(text='Facial and AD', x=0.10, y=0.5, font_size=18, showarrow=False),
                 dict(text='Beard and Hair', x=0.50, y=0.5, font_size=18, showarrow=False),
                 dict(text='Beard and Dye', x=0.90, y=0.5, font_size=18, showarrow=False)])
fig.show()
```