5ffiikspi

April 29, 2024

[ ]:
```
import pandas as pd
import numpy as np
import plotly.express as px
```

1 Exploring Dataset

[ ]:
```
# dataset
data = pd.read_csv("/content/Market_Basket_Optimisation (2).csv")
# printing the shape of the dataset
data.shape
```

[ ]:
```
(7500, 20)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7500 entries, 0 to 7499
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   shrimp             7500 non-null   object
 1   almonds            5746 non-null   object
 2   avocado            4388 non-null   object
 3   vegetables mix     3344 non-null   object
 4   green grapes       2528 non-null   object
 5   whole weat flour   1863 non-null   object
 6   yams               1368 non-null   object
 7   cottage cheese     980 non-null    object
 8   energy drink       653 non-null    object
 9   tomato juice       394 non-null    object
 10  low fat yogurt     255 non-null    object
 11  green tea          153 non-null    object
 12  honey              86 non-null     object
 13  salad              46 non-null     object
 14  mineral water      24 non-null     object
 15  salmon             7 non-null      object
 16  antioxydant juice  3 non-null      object
```

```
 17  frozen smoothie    3 non-null      object
 18  spinach            2 non-null      object
 19  olive oil          0 non-null      float64
dtypes: float64(1), object(19)
memory usage: 1.1+ MB
```

[ ]:
```
data.tail()
```

[ ]:
```
        shrimp            almonds      avocado  vegetables mix green grapes  \
7495    butter         light mayo  fresh bread             NaN          NaN
7496   burgers  frozen vegetables         eggs    french fries    magazines
7497   chicken                NaN          NaN             NaN          NaN
7498  escalope          green tea          NaN             NaN          NaN
7499      eggs    frozen smoothie  yogurt cake  low fat yogurt          NaN

     whole weat flour yams cottage cheese energy drink tomato juice  \
7495              NaN  NaN            NaN          NaN          NaN
7496        green tea  NaN            NaN          NaN          NaN
7497              NaN  NaN            NaN          NaN          NaN
7498              NaN  NaN            NaN          NaN          NaN
7499              NaN  NaN            NaN          NaN          NaN

     low fat yogurt green tea honey salad mineral water salmon  \
7495            NaN       NaN   NaN   NaN           NaN    NaN
7496            NaN       NaN   NaN   NaN           NaN    NaN
7497            NaN       NaN   NaN   NaN           NaN    NaN
7498            NaN       NaN   NaN   NaN           NaN    NaN
7499            NaN       NaN   NaN   NaN           NaN    NaN

     antioxydant juice frozen smoothie spinach  olive oil
7495               NaN             NaN     NaN        NaN
7496               NaN             NaN     NaN        NaN
7497               NaN             NaN     NaN        NaN
7498               NaN             NaN     NaN        NaN
7499               NaN             NaN     NaN        NaN
```

[ ]:
```
transaction = []
for i in range(0, data.shape[0]):
    for j in range(0, data.shape[1]):
        transaction.append(data.values[i,j])
```

[ ]:
```
# converting to numpy array
transaction = np.array(transaction)
```

[ ]:
```
#  Transform Them a Pandas DataFrame
df = pd.DataFrame(transaction, columns=["items"])
# Put 1 to Each Item For Making Countable Table, to be able to perform Group By
```

```
df["incident_count"] = 1
#  Delete NaN Items from Dataset
indexNames = df[df['items'] == "nan" ].index
df.drop(indexNames , inplace=True)
# Making a New Appropriate Pandas DataFrame for Visualizations
df_table = df.groupby("items").sum().sort_values("incident_count",␣
 ↪ascending=False).reset_index()
#  Initial Visualizations
df_table.head(10).style.background_gradient(cmap='coolwarm')
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x7c6e2d26aa70>
```

The output shows that mineral water has been purchased more frequently than other products.

[ ]:
```
# Create a bar chart using Plotly Express
fig = px.bar(df_table.head(30), x='items', y='incident_count',
             title='Item Incident Counts',
             labels={'items': 'Items', 'incident_count': 'Incident Count'},
             color='incident_count',
             color_continuous_scale='sunsetdark')

# Customize the layout if needed
fig.update_layout(
    xaxis=dict(tickangle=-45),
    yaxis=dict(title='Incident Count'),
    coloraxis_colorbar=dict(title='Count', tickformat=','),
)

# Show the bar chart
fig.show()
```

A Barcharting is a method for displaying hierarchical data using nested figures. We can use a barchart to visualize all the items from our dataset more interactive.

2 Data Preprocessing

Before getting the most frequent itemsets, the dataset needs to be transformed into a True – False matrix where rows are transactions and columns are products.

[ ]:
```
transaction = []
for i in range(0,data.shape[0]):
    transaction.append([str(data.values[i,j])for j in range(0,data.shape[1])])
```

[ ]:
```
# importing the required module
from mlxtend.preprocessing import TransactionEncoder
# initializing the transactionEncoder
te = TransactionEncoder()
```

```
te_ary = te.fit(transaction).transform(transaction)
dataset = pd.DataFrame(te_ary, columns=te.columns_)
# dataset after encoded
dataset
```

[ ]:

	asparagus	almonds	antioxydant juice	asparagus	avocado	babies food	\
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	True	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
…	…	…	…	…	…	…	
7495	False	False	False	False	False	False	
7496	False	False	False	False	False	False	
7497	False	False	False	False	False	False	
7498	False	False	False	False	False	False	
7499	False	False	False	False	False	False	

	bacon	barbecue sauce	black tea	blueberries	…	turkey	\
0	False	False	False	False	…	False	
1	False	False	False	False	…	False	
2	False	False	False	False	…	True	
3	False	False	False	False	…	False	
4	False	False	False	False	…	False	
…	…	…	…	…	…	…	
7495	False	False	False	False	…	False	
7496	False	False	False	False	…	False	
7497	False	False	False	False	…	False	
7498	False	False	False	False	…	False	
7499	False	False	False	False	…	False	

	vegetables mix	water spray	white wine	whole weat flour	\
0	False	False	False	False	
1	False	False	False	False	
2	False	False	False	False	
3	False	False	False	False	
4	False	False	False	False	
…	…	…	…	…	
7495	False	False	False	False	
7496	False	False	False	False	
7497	False	False	False	False	
7498	False	False	False	False	
7499	False	False	False	False	

	whole wheat pasta	whole wheat rice	yams	yogurt cake	zucchini
0	False	False	False	False	False
1	False	False	False	False	False

```
2                False             False  False        False     False
3                False              True  False        False     False
4                False             False  False        False     False
…                …                 …      …            …         …
7495             False             False  False        False     False
7496             False             False  False        False     False
7497             False             False  False        False     False
7498             False             False  False        False     False
7499             False             False  False         True     False

[7500 rows x 121 columns]
```

We have 121 columns/features at the moment. Extracting the most frequent itemsets from 121 features would be compelling. So, we will start with the Top 50 items.

[ ]:
```
# select top 50 items
first50 = df_table["items"].head(50).values
# Extract Top50
dataset = dataset.loc[:,first50]
# shape of the dataset
dataset.shape
```

[ ]:
```
(7500, 50)
```

3 Apriori Algorithm

[ ]:
```
# importing the required module
from mlxtend.frequent_patterns import apriori, association_rules

# Extracting the most frequest itemsets via Mlxtend.
# The length column has been added to increase ease of filtering.
frequent_itemsets = apriori(dataset, min_support=0.01, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:␣
 ↪len(x))
# printing the frequent itemset
frequent_itemsets
```

[ ]:
```
      support                                itemsets  length
0    0.238267                         (mineral water)       1
1    0.179733                                 (eggs)       1
2    0.174133                            (spaghetti)       1
3    0.170933                          (french fries)       1
4    0.163867                             (chocolate)       1
..        …                                       …     …
229  0.010933   (ground beef, mineral water, chocolate)       3
230  0.011067        (milk, ground beef, mineral water)       3
```

```
231  0.011067  (frozen vegetables, milk, mineral water)       3
232  0.010533              (spaghetti, eggs, chocolate)       3
233  0.010933              (spaghetti, milk, chocolate)       3

[234 rows x 3 columns]
```

The output shows that mineral water is the dataset's most frequently occurring item. For further experiment, we can print out all items with a length of 2, and the minimum support is more than 0.05.

[ ]:
```
# printing the frequntly items
frequent_itemsets[ (frequent_itemsets['length'] == 2) &
                   (frequent_itemsets['support'] >= 0.05) ]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning:

`should_run_async` will not call `transform_cell` automatically in the future.
Please pass the result to `transformed_cell` argument and any exception that
happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and
above.
```

[ ]:
```
     support                    itemsets  length
50  0.050933       (eggs, mineral water)       2
51  0.059733  (spaghetti, mineral water)       2
53  0.052667  (mineral water, chocolate)       2
```

The output shows that the eggs and mineral water combination are the most frequently occurring items when the length of the itemset is two.

Similarly, we can find the most frequently occurring items when the itemset length is 3:

[ ]:
```
# printing the frequntly items with length 3
frequent_itemsets[ (frequent_itemsets['length'] == 3) ].head(3)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning:

`should_run_async` will not call `transform_cell` automatically in the future.
Please pass the result to `transformed_cell` argument and any exception that
happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and
above.
```

[ ]:
```
      support                          itemsets  length
217  0.014267  (eggs, spaghetti, mineral water)       3
218  0.013467  (eggs, mineral water, chocolate)       3
219  0.013067       (eggs, milk, mineral water)       3
```

4 Further Association Rules

```
[ ]: #  We set our metric as "Lift" to define whether antecedents & consequents are␣
      ↪dependent our not
     rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.2)
     rules["antecedents_length"] = rules["antecedents"].apply(lambda x: len(x))
     rules["consequents_length"] = rules["consequents"].apply(lambda x: len(x))
     rules.sort_values("lift",ascending=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning:

`should_run_async` will not call `transform_cell` automatically in the future.
Please pass the result to `transformed_cell` argument and any exception that
happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and
above.
```

```
[ ]:                      antecedents                   consequents  \
     218               (herb & pepper)               (ground beef)
     219                 (ground beef)               (herb & pepper)
     293                 (ground beef)  (spaghetti, mineral water)
     292    (spaghetti, mineral water)               (ground beef)
     311                   (olive oil)  (spaghetti, mineral water)
     ..                           …                           …
     60                         (eggs)            (low fat yogurt)
     122                    (escalope)              (french fries)
     123                (french fries)                  (escalope)
     165                      (shrimp)                 (green tea)
     164                   (green tea)                    (shrimp)

          antecedent support  consequent support   support  confidence      lift  \
     218            0.049467            0.098267  0.016000    0.323450  3.291555
     219            0.098267            0.049467  0.016000    0.162822  3.291555
     293            0.098267            0.059733  0.017067    0.173677  2.907540
     292            0.059733            0.098267  0.017067    0.285714  2.907540
     311            0.065733            0.059733  0.010267    0.156187  2.614731
     ..                  …                   …         …           …         …
     60             0.179733            0.076400  0.016800    0.093472  1.223453
     122            0.079333            0.170933  0.016400    0.206723  1.209376
     123            0.170933            0.079333  0.016400    0.095944  1.209376
     165            0.071333            0.132000  0.011333    0.158879  1.203625
     164            0.132000            0.071333  0.011333    0.085859  1.203625

          leverage  conviction  zhangs_metric  antecedents_length  \
     218  0.011139    1.332841       0.732423                   1
     219  0.011139    1.135402       0.772060                   1
```

```
293  0.011197    1.137893       0.727562                   1
292  0.011197    1.262427       0.697745                   2
311  0.006340    1.114306       0.661001                   1
..        …          …             …                     …
60   0.003068    1.018832       0.222661                   1
122  0.002839    1.045116       0.188046                   1
123  0.002839    1.018373       0.208822                   1
165  0.001917    1.031956       0.182171                   1
164  0.001917    1.015890       0.194904                   1

     consequents_length
218                   1
219                   1
293                   2
292                   1
311                   2
..                  …
60                    1
122                   1
123                   1
165                   1
164                   1

[350 rows x 12 columns]
```