

**NAAN MUDHALVAN – IBM SKILL  
ARTIFICIAL INTELLIGENCE  
GROUP PROJECT**

**Project Title: Market Basket Insight  
Phase 3 Submission**

<b>S.NO</b>	<b>GROUP MEMBERS NAME</b>	<b>NM ID</b>	<b>E-MAIL ID</b>
1	DURAI BHUVANESH.CM	au820321106013	cmduraicmdurai12@gmail.com
2	SUJITHKUMAR.R	au820321106037	sujithkumarrao333@gmail.com
3	NARENDHIRAN.R	au820321106027	naveenrc430@gmail.com
4	VIMAL.M	au820321106039	mm4795231@gmail.com
5	KARUNAMOORTHY.K	au820321106022	karunamoorthy8012@gmail.com

# MARKET BASKET ANALYSIS USING PYTHON

**Market basket analysis** is a data mining technique used by retailers to increase sales by better understanding customer purchasing patterns. It involves analyzing large data sets, such as purchase history, to reveal product groupings, as well as products that are likely to be purchased together [1].

E.g. the rule **{cucumbers, tomatoes} -> {sunflower oil}** found in the sales data of a supermarket would indicate that if a customer buys cucumbers and tomatoes together, they are likely to also buy sunflower oil.

## 1. Import Libraries

For market basket analysis I'm going to use [mlxtend](#). For other purposes (reading data, working with data, visualizing data) I'll use all well-known libraries like numpy, pandas etc.

In [1]:

```
import numpy as np
import pandas as pd
import squarify
import matplotlib.pyplot as plt

# for market basket analysis
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder
```

## 2. Read data

In [2]:

```
df = pd.read_excel('C:\marketbasket\Assignment1_Data.xlsx ', header = None)
```

In [3]:

```
df.head(5) # Looking for the first 5 rows in dataset
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole wheat	yeast	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salsad	mineral water	salmmon	antioxidant juice	frozen smoothie	spinach	olive oil

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
					es	t fl o ur		se		ce	rt	a								
1	bu rge rs	me atb alls	eg gs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	ch ut ne y	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	tur ke y	avo cad o	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mi ne ral wa ter	mil k	en erg y bar	who le whe at rice	gre en tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

### 3. Visualize data

Here I decided to count all unique values through all columns and build some visualitions. E.g. if we have 5 'almonds' in first column, 3 'almonds' in second column etc, so, will have 8 'almonds' in total.

In [4]:

```
df_res = pd.DataFrame()
for i in range(len(df.columns)):
    df_res = df_res.append(df[i].value_counts())
```

In [5]:

```
df_res.head(3)
```

```
Out[5]:
```

	almonds	antioxidant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	bodyspray	.	.	.	whole wheat flour	whole wheat pasta	whole wheat rice	yams	yo-yurt cake	tea	water spray	zucchini	napkins	asparagus
0	11.0	18.0	3.0	57.0	5.0	6.0	3.0	9.0	4.0	1.0	.	.	.	8.0	95.0	47.0	24.0	31.0	NaN	NaN	NaN	NaN	NaN
1	29.0	10.0	2.0	64.0	5.0	8.0	9.0	31.0	8.0	13.0	.	.	.	5.0	68.0	92.0	25.0	38.0	5.0	1.0	10.0	NaN	NaN
2	35.0	12.0	5.0	46.0	4.0	12.0	18.0	15.0	13.0	14.0	.	.	.	8.0	33.0	69.0	24.0	32.0	4.0	1.0	2.0	NaN	NaN

3 rows x 120 columns

```
In [6]:
```

```
df_sum = df_res.sum()
```

```
df_sum = df_sum.sort_values(ascending=False)
```

```
In [7]:
```

```
df_sum
```

```
Out[7]:
```

```
mineral water    1788.0
eggs             1348.0
spaghetti        1306.0
french fries     1282.0
chocolate        1230.0
...
```

```

bramble          14.0
cream            7.0
napkins          5.0
water spray      3.0
  asparagus      1.0
Length: 120, dtype: float64

```

After counting all values through all columns, we can build a **frequency plot**.

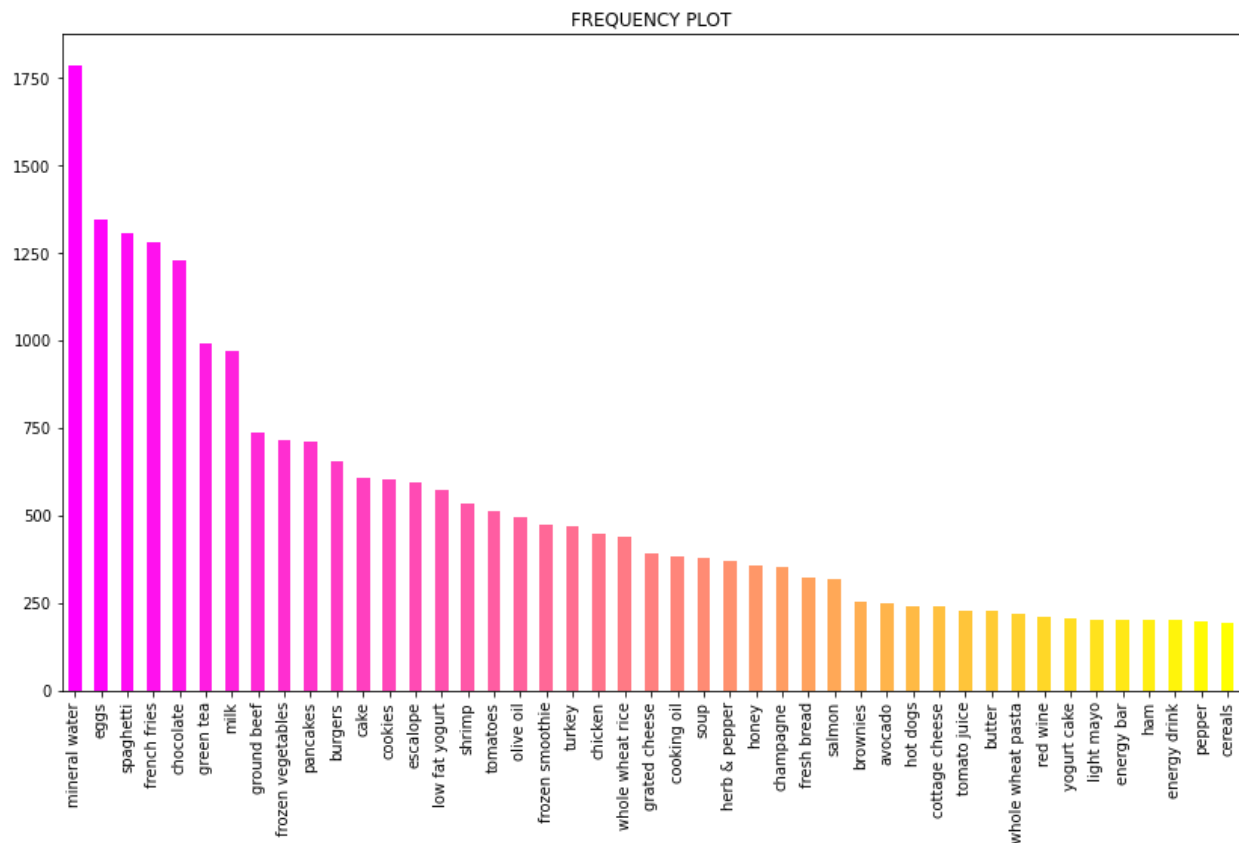
In [8]:

linkcode

```

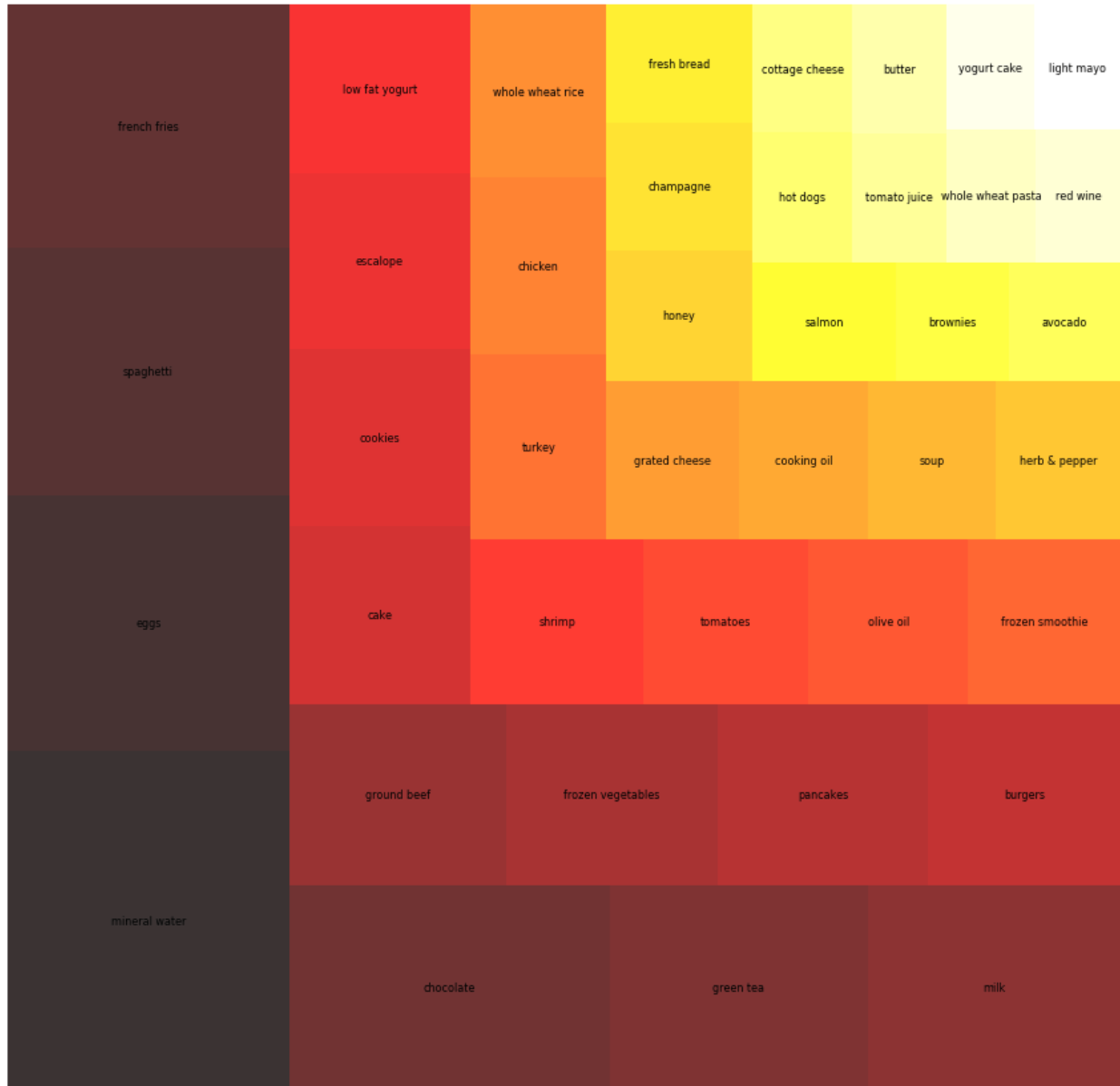
plt.figure(figsize=(14,8))
plt.title("FREQUENCY PLOT")
cnt = 45 # plot only first 'cnt' values
color = plt.cm.spring(np.linspace(0, 1, cnt))
df_sum.head(cnt).plot.bar(color = color)
plt.xticks(rotation = 'vertical')
plt.grid(False)
plt.axis('on')
plt.show()

```



Also we can frequency plot, but in the form of **heat map**:

```
In [9]:
plt.figure(figsize=(15,15))
cnt = 40 # plot only first 'cnt' values
color = plt.cm.hot(np.linspace(0, 1, cnt))
df_part = df_sum.head(cnt)
squarify.plot(sizes = df_part.values, label = df_part.index, alpha=.8, color = color,
text_kwargs={'fontsize':8})
plt.axis('off')
plt.show()
```



## 4. Transform data

Before converting dataset to transaction view, we should convert pandas-data to list-data and then list-data to numpy-data.

In [10]:

```
# making each customers shopping items an identical list
arr = []
for i in range(df.shape[0]):
    arr.append([str(df.values[i,j]) for j in range(df.shape[1])])
```

```
arr = np.array(arr)
print(arr.shape)
(7501, 20)
```

And now we can convert our dataset to **transaction view**.

In [11]:

```
te = TransactionEncoder()
data = te.fit_transform(arr)
data = pd.DataFrame(data, columns = te.columns_)
print(data.shape)
(7501, 121)
```

And then we can check the results:

In [12]:

```
data.head(3)
```

Out[12]:

	asp ara gus	al m on ds	anti oxy dan t juic e	asp ara gus	av oc ad o	b a bi es fo od	b ac on	bar be cu e sau ce	bl a c k t ea	blu ebe rrie s	. . .	tu rk ey	veg eta ble s mix	w at er s pra y	w hi te win e	w h ole we at fl o ur	w h ole we h eat pas ta	w h ole we h eat rice	y a ms	yo g ur t ca ke	zu cc hi ni
0	False	True	True	False	True	False	False	False	False	False	. . .	False	True	False	False	True	False	False	True	False	False

	asp ara gus	al m on ds	anti oxy dan t juic e	asp ara gus	av oc ad o	b a bi es fo od	b ac on	bar be cu e sau ce	bl a c k t e a	blu ebe rrie s	.	tu rk ey	veg eta ble s mix	w at er s pr a y	w hi te w in e	w h ol e w he at fl o ur	w h ol e w he at p as ta	w h ol e w he at ri ce	y a m s	yo g ur t ca ke	zu cc hi ni
1	Fal se	Fal se	Fals e	Fal se	Fal se	Fa ls e	F al se	Fal se	F al s e	Fals e	.	Fa ls e	Fals e	F al se	F al s e	F al se	Fa ls e	Fa ls e	F al s e	Fa ls e	Fal se
2	Fal se	Fal se	Fals e	Fal se	Fal se	Fa ls e	F al se	Fal se	F al s e	Fals e	.	Fa ls e	Fals e	F al se	F al s e	F al se	Fa ls e	Fa ls e	F al s e	Fa ls e	Fal se

3 rows x 121 columns

As we can see, we have '**nan**' column, so we should drop it, because these are just 'not a number' values, which were only empty cells in the original dataset.

In [13]:

```
data = data.drop(columns=['nan'])
data.head(3)
```

Out[13]:



	asp ara gus	al m on ds	anti oxy dan t juic e	asp ara gus	av oc ad o	b a bi es fo od	b ac on	bar be cu e sau ce	bl a c k t e a	blu ebe rrie s	. . .	tu rk ey	veg eta ble s mix	w at er s pr a y	w hi te w in e	w h ol e w he at fl o ur	w h ol e w he at pas ta	w h ol e w he at rice	y a ms	yo g urt ca ke	zu cc hi ni
0	Fal se	Tr ue	Tru e	Fal se	Tr ue	Fal se	Fal se	Fal se	Fal se	Fals e	. . .	Fal se	Tru e	Fal se	Fal se	Tr u e	Fal se	Fal se	Tr u e	Fal se	Fal se
1	Fal se	Fal se	Fals e	Fal se	Fal se	Fal se	Fal se	Fal se	Fal se	Fals e	. . .	Fal se	Fals e	Fal se	Fal se	Fal se	Fal se	Fal se	Fal se	Fal se	Fal se
2	Fal se	Fal se	Fals e	Fal se	Fal se	Fal se	Fal se	Fal se	Fal se	Fals e	. . .	Fal se	Fals e	Fal se	Fal se	Fal se	Fal se	Fal se	Fal se	Fal se	Fal se

3 rows x 120 columns

linkcode

## 5. Analyze data with apriori rule

Apriori algorithm assumes that any subset of a frequent itemset must be frequent. Its the algorithm behind Market Basket Analysis. Say, a transaction containing {Grapes, Apple, Mango} also contains {Grapes, Mango}. So, according to the principle of Apriori, if {Grapes, Apple, Mango} is frequent, then {Grapes, Mango} must also be frequent .

**Support:** Its the default popularity of an item. In mathematical terms, the support of item A is nothing but the ratio of transactions involving A to the total number of transactions.  $Support(Grapes) = (Transactions\ involving\ Grapes)/(Total\ transaction)$  .

**Confidence:** Likelihood that customer who bought both A and B. Its divides the number of transactions involving both A and B by the number of transactions involving B.  $Confidence(A \Rightarrow B) = (Transactions\ involving\ both\ A\ and\ B) / (Transactions\ involving\ only\ A)$  .

$$\begin{array}{l}
 \text{Rule } X \Rightarrow Y \begin{cases} \text{Support} = \frac{Frequency(X,Y)}{N} \\ \text{Confidence} = \frac{Frequency(X,Y)}{Frequency(X)} \\ \text{Lift} = \frac{Support}{Support(X) * Support(Y)} \end{cases}
 \end{array}$$

So, now we are going to use **apriori rule** to find some dependencies. [Here](#) you can read more about it.

In [14]:

```
freq_rules = apriori(data, min_support = 0.01, use_colnames = True)
freq_rules
```

Out[14]:

	support	itemsets
0	0.020397	(almonds)
1	0.033329	(avocado)
2	0.010799	(barbecue sauce)
3	0.014265	(black tea)
4	0.011465	(body spray)

	support	itemsets
...	...	...
252	0.011065	(ground beef, milk, mineral water)
253	0.017064	(spaghetti, ground beef, mineral water)
254	0.015731	(spaghetti, milk, mineral water)
255	0.010265	(spaghetti, olive oil, mineral water)
256	0.011465	(pancakes, spaghetti, mineral water)

257 rows x 2 columns

So, here we can see all rules, that have minimum support 0.01. If you need **rules with certain length**, we can filter the results:

In [15]:

```
freq_rules['length'] = freq_rules['itemsets'].apply(lambda x: len(x)) # adding 'length' column
```

freq\_rules

Out[15]:

	support	itemsets	length
0	0.020397	(almonds)	1
1	0.033329	(avocado)	1

	support	itemsets	length
2	0.010799	(barbecue sauce)	1
3	0.014265	(black tea)	1
4	0.011465	(body spray)	1
...	...	...	...
252	0.011065	(ground beef, milk, mineral water)	3
253	0.017064	(spaghetti, ground beef, mineral water)	3
254	0.015731	(spaghetti, milk, mineral water)	3
255	0.010265	(spaghetti, olive oil, mineral water)	3
256	0.011465	(pancakes, spaghetti, mineral water)	3

257 rows x 3 columns

In [16]:

```
mask = freq_rules['length'] > 1 # creating mask for filtering with certain condition
filtered_freq_rules = freq_rules.loc[mask] # applying mask
filtered_freq_rules # printing the filtering result
```

Out[16]:

	support	itemsets	length
75	0.011598	(avocado, mineral water)	2
76	0.011465	(burgers, cake)	2
77	0.017064	(burgers, chocolate)	2
78	0.028796	(burgers, eggs)	2
79	0.021997	(burgers, french fries)	2
...	...	...	...
252	0.011065	(ground beef, milk, mineral water)	3
253	0.017064	(spaghetti, ground beef, mineral water)	3
254	0.015731	(spaghetti, milk, mineral water)	3
255	0.010265	(spaghetti, olive oil, mineral water)	3
256	0.011465	(pancakes, spaghetti, mineral water)	3

182 rows x 3 columns

So, in the end we can see all rules, which have certain length and which have certain minimum support.

linkcode

## 6. Conclusion

More and more organizations are discovering ways of using market basket analysis to gain useful insights into associations and hidden relationships. As industry leaders continue to explore the technique's value, a predictive version of market basket analysis is making in-roads across many sectors in an effort to identify sequential purchases.

Thank you for reading my new article! Hope, you liked it and it was interesting for you! There are some more my articles: