KDD- Experiment 7

Fp Growth Algorithm

- **SIA VASHIST**
- PRN: 20190802107

Dataset used: Market Basket Optimisation

Libraries used:

Pandas | Numpy | matplotlib | mlxtend | plotly | squarify

In [1]:
```
# %pip install mlxtend --upgrade
```

In [2]:
```
# importing module
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#Filter warnings
import warnings
warnings.filterwarnings('ignore', category=DeprecationWarning)

# dataset
dataset = pd.read_csv(r'C:\sia\Market_Basket_Optimisation.csv', header=None)
```

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

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 weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	frozen smoothie	spinach	olive oil
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	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	NaN	NaN	NaN	NaN
3	turkey	avocado	NaN	NaN	NaN	NaN	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	NaN	NaN	NaN	NaN

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

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

In [5]:
```
# importing module
import numpy as np
# Gather All Items of Each Transactions into Numpy Array
transaction = []
for i in range(0, dataset.shape[0]):
    for j in range(0, dataset.shape[1]):
        transaction.append(dataset.values[i,j])
# converting to numpy array
transaction = np.array(transaction)
print(transaction)
```

```
['shrimp' 'almonds' 'avocado' ... 'nan' 'nan' 'nan']
```

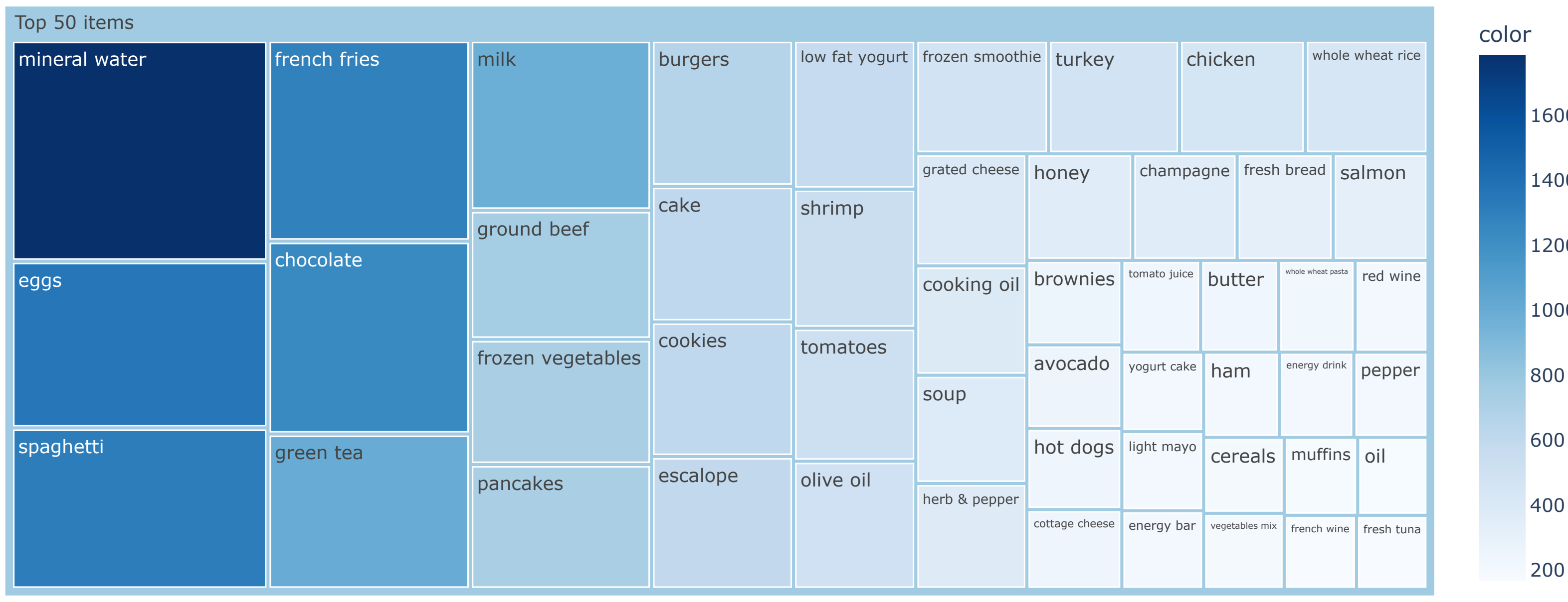
In [6]:
```
#  Transform Them a Pandas DataFrame
df = pd.DataFrame(transaction, columns=["items"])
# Put 1 to Each Item For Making Countable Table, to be able to perform Group By
df["incident_count"] = 1
#  Delete NaN Items from Dataset
indexNames = df[df['items'] == "nan" ].index
df.drop(indexNames , 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(5).style.background_gradient(cmap='Blues')
```

Out[6]:

	items	incident_count
0	mineral water	1788
1	eggs	1348
2	spaghetti	1306
3	french fries	1282
4	chocolate	1230

In [7]:
```
# importing required module
import plotly.express as px
# to have a same origin
df_table["all"] = "Top 50 items"
# creating tree map using plotly
fig = px.treemap(df_table.head(50), path=['all', "items"], values='incident_count',
                  color=df_table["incident_count"].head(50), hover_data=['items'],
                  color_continuous_scale='Blues',
                )
# ploting the treemap
fig.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  df_all_trees = df_all_trees.append(df_tree, ignore_index=True)
C:\Users\HP\anaconda3\lib\site-packages\plotly\express\_core.py:1637: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  df_all_trees = df_all_trees.append(df_tree, ignore_index=True)
```



> ## Observations:
>
> The size of each rectangle in the treemap represents the frequency of the item in the dataset. Larger rectangles represent items that appear more frequently.
>
> The color of each rectangle in the treemap represents the importance of the item in the dataset. Darker colors represent items that are more important.
>
> The treemap visualization is a useful tool for identifying patterns and trends in large datasets. It allows us to quickly identify the most frequent items in the dataset and the relationships between different items.

In [8]:

```
# Transform Every Transaction to Seperate List & Gather Them into Numpy Array
transaction = []
for i in range(dataset.shape[0]):
    transaction.append([str(dataset.values[i,j]) for j in range(dataset.shape[1])])
# creating the numpy array of the transactions
transaction = np.array(transaction)
# 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.head()
```

Out[8]:

	asparagus	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	...	turkey	vegetables mix	water spray	white wine	whole weat flour	whole wheat pasta	whole wheat rice	yams	yogurt cake	zucchini
0	False	True	True	False	True	False	False	False	False	False	...	False	True	False	False	True	False	False	True	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	True	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	True	False	False	False

5 rows × 121 columns

In [9]:

```
# select top 30 items
first30 = df_table["items"].head(30).values
# Extract Top 30
dataset = dataset.loc[:,first30]
# shape of the dataset
dataset.shape
```

Out[9]:

```
(7501, 30)
```

In [10]:

```
#Importing Libraries
from mlxtend.frequent_patterns import fpgrowth
#running the fpgrowth algorithm
res=fpgrowth(dataset,min_support=0.05, use_colnames=True)
# printing top 10
res.head(10)
```

Out[10]:

	support	itemsets
0	0.238368	(mineral water)
1	0.132116	(green tea)
2	0.076523	(low fat yogurt)
3	0.071457	(shrimp)
4	0.065858	(olive oil)
5	0.063325	(frozen smoothie)
6	0.179709	(eggs)
7	0.087188	(burgers)
8	0.062525	(turkey)
9	0.129583	(milk)

In [11]:

```
# importing required module
from mlxtend.frequent_patterns import association_rules
# creating asssociation rules
res=association_rules(res, metric="lift", min_threshold=1)
# printing association rules
res
```

Out[11]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(eggs)	(mineral water)	0.179709	0.238368	0.050927	0.283383	1.188845	0.008090	1.062815
1	(mineral water)	(eggs)	0.238368	0.179709	0.050927	0.213647	1.188845	0.008090	1.043158
2	(spaghetti)	(mineral water)	0.174110	0.238368	0.059725	0.343032	1.439085	0.018223	1.159314
3	(mineral water)	(spaghetti)	0.238368	0.174110	0.059725	0.250559	1.439085	0.018223	1.102008
4	(chocolate)	(mineral water)	0.163845	0.238368	0.052660	0.321400	1.348332	0.013604	1.122357
5	(mineral water)	(chocolate)	0.238368	0.163845	0.052660	0.220917	1.348332	0.013604	1.073256

In [12]:

```
# Sort values based on confidence
res.sort_values("confidence",ascending=False)
```

Out[12]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(spaghetti)	(mineral water)	0.174110	0.238368	0.059725	0.343032	1.439085	0.018223	1.159314
4	(chocolate)	(mineral water)	0.163845	0.238368	0.052660	0.321400	1.348332	0.013604	1.122357
0	(eggs)	(mineral water)	0.179709	0.238368	0.050927	0.283383	1.188845	0.008090	1.062815
3	(mineral water)	(spaghetti)	0.238368	0.174110	0.059725	0.250559	1.439085	0.018223	1.102008
5	(mineral water)	(chocolate)	0.238368	0.163845	0.052660	0.220917	1.348332	0.013604	1.073256
1	(mineral water)	(eggs)	0.238368	0.179709	0.050927	0.213647	1.188845	0.008090	1.043158

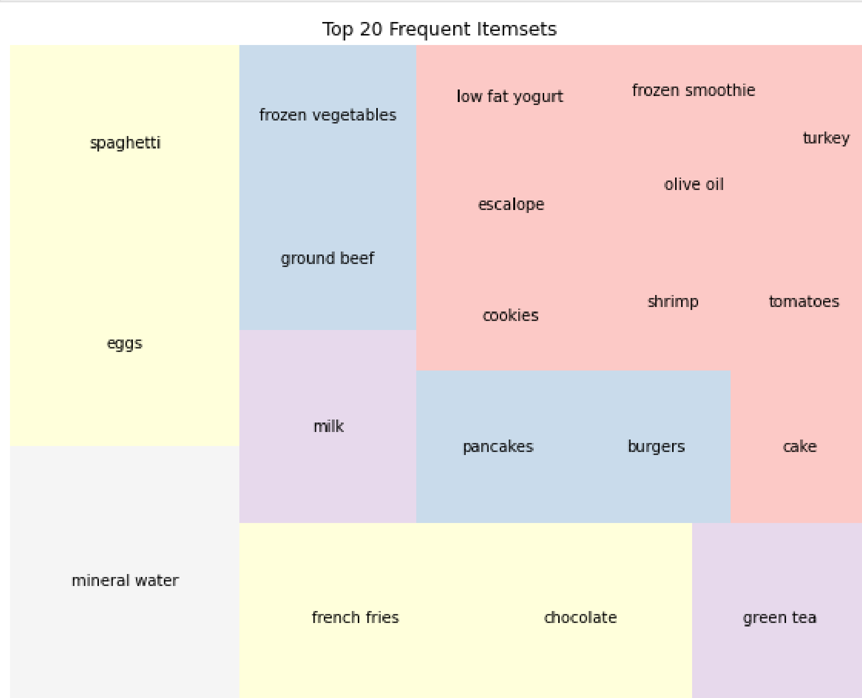
In [13]:

```
# Convert categorical variables to binary variables using one-hot encoding
df2 = pd.get_dummies(dataset)

# Run the FP-Growth algorithm
frequent_itemsets = fpgrowth(df2, min_support=0.05, use_colnames=True)
```

In [14]:

```
import squarify
# Create the treemap
top_itemsets = frequent_itemsets.sort_values('support', ascending=False).head(20)
labels = [", ".join(itemset) for itemset in top_itemsets['itemsets']]
sizes = top_itemsets['support'].tolist()
cmap = plt.get_cmap('Pastel1')
norm = plt.Normalize(min(sizes), max(sizes))
colors = cmap(norm(sizes))
fig, ax = plt.subplots()
fig.set_size_inches(10, 8)
squarify.plot(sizes=sizes, label=labels, color=colors, alpha=.7, ax=ax)
plt.title('Top 20 Frequent Itemsets')
plt.axis('off')
plt.show()
```



Conclusion:

> FP-Growth is a powerful algorithm for mining frequent itemsets in large datasets. It is more efficient than Apriori for datasets with a large number of transactions or items. The treemap visualization is a useful tool for identifying the most frequent itemsets and their support values. The colormap can help to quickly identify the most and least frequent itemsets in the dataset. The labels provide additional information about the itemsets, including the individual items that make up the itemset.