Comparision between Apriori and FP Growth

Imports

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
import seaborn as sns
import plotly.express as px
from pandas.api.types import CategoricalDtype
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth
```

Reading the Groceries Dataset into python using Pandas Library and viewing the first 5 elements of the Dataset

In [3]:

```
groceries=pd.read_csv("D:\Groceries_dataset.csv")
groceries.head()
```

```
<>:1: DeprecationWarning: invalid escape sequence '\G'
<>:1: DeprecationWarning: invalid escape sequence '\G'
C:\Users\hp\AppData\Local\Temp\ipykernel_22540\4165234428.py:1: DeprecationWarning: invalid escape sequence '\G'
  groceries=pd.read_csv("D:\Groceries_dataset.csv")
```

Out[3]:

	Member_number	Date	itemDescription
0	1808	21-07-2015	tropical fruit
1	2552	05-01-2015	whole milk
2	2300	19-09-2015	pip fruit
3	1187	12-12-2015	other vegetables
4	3037	01-02-2015	whole milk

Getting an idea on the dataset by looking into the info

We can see the number of Columns ,Name,count of the data and the Data type of each columns

In [4]:

```
groceries.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38765 entries, 0 to 38764
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Member_number    38765 non-null  int64 
 1   Date             38765 non-null  object
 2   itemDescription  38765 non-null  object
dtypes: int64(1), object(2)
memory usage: 908.7+ KB
```

Checking wheather the dataset contains null values or not

In [5]:

```
groceries.isna().sum()
```

Out[5]:

```
Member_number      0
Date               0
itemDescription    0
dtype: int64
```

Converting the date from object type to datetime64 type

In [6]:

```
groceries['Date']=pd.to_datetime(groceries['Date'])
groceries['Member_number']= groceries['Member_number'].astype('object')
groceries.info()
```

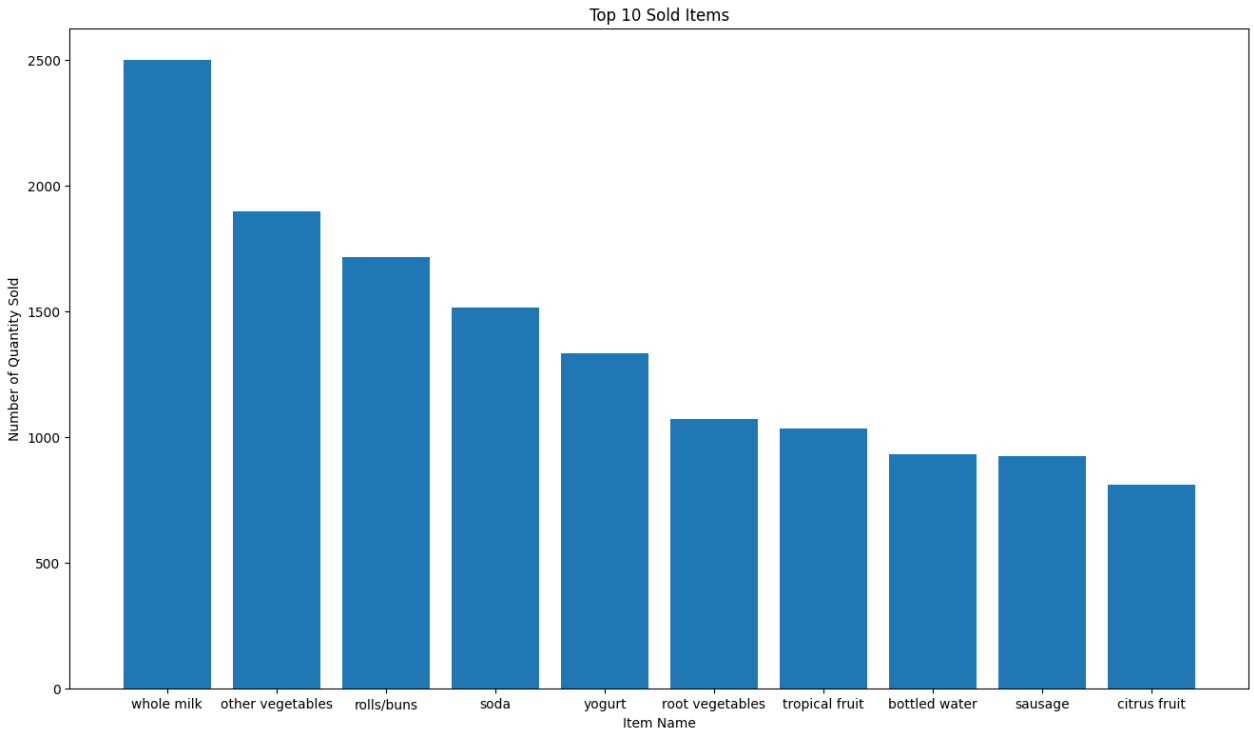
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38765 entries, 0 to 38764
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Member_number    38765 non-null  object
 1   Date             38765 non-null  datetime64[ns]
 2   itemDescription  38765 non-null  object
dtypes: datetime64[ns](1), object(2)
memory usage: 908.7+ KB
```

```
C:\Users\hp\anaconda3\envs\tf1\lib\site-packages\pandas\core\tools\datetimes.py:1047: UserWarning: Parsing '21-0
7-2015' in DD/MM/YYYY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\hp\anaconda3\envs\tf1\lib\site-packages\pandas\core\tools\datetimes.py:1047: UserWarning: Parsing '19-0
9-2015' in DD/MM/YYYY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\hp\anaconda3\envs\tf1\lib\site-packages\pandas\core\tools\datetimes.py:1047: UserWarning: Parsing '14-0
2-2015' in DD/MM/YYYY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
```

Finding the first top 10 highest sold groceries items

In [7]:

```
Item_distr = groceries.groupby(by = "itemDescription").size().reset_index(name='Frequency').sort_values(by = 'Frequency',ascendin
bars = Item_distr["itemDescription"]
height = Item_distr["Frequency"]
x_pos = np.arange(len(bars))
plt.figure(figsize=(16,9))
plt.bar(x_pos, height)
plt.title("Top 10 Sold Items")
plt.xlabel("Item Name")
plt.ylabel("Number of Quantity Sold")
plt.xticks(x_pos, bars)
plt.show()
```



In [8]:

```
df_date=groceries.set_index(['Date'])
df_date
```

Out[8]:

Date	Member_number	itemDescription
2015-07-21	1808	tropical fruit
2015-05-01	2552	whole milk
2015-09-19	2300	pip fruit
2015-12-12	1187	other vegetables
2015-01-02	3037	whole milk
...	...	...
2014-08-10	4471	sliced cheese
2014-02-23	2022	candy
2014-04-16	1097	cake bar
2014-03-12	1510	fruit/vegetable juice
2014-12-26	1521	cat food

38765 rows × 2 columns

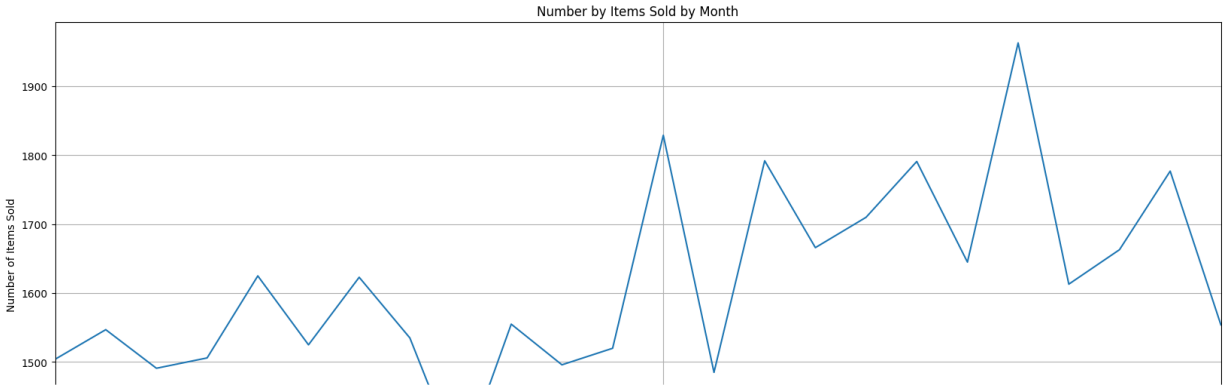
No of items sold by month

In [9]:

```
df_date.resample("M")['itemDescription'].count().plot(figsize = (20,8), grid = True, title = "Number by Items Sold by Month").set
```

Out[9]:

```
[Text(0.5, 0, 'Date'), Text(0, 0.5, 'Number of Items Sold')]
```



Converting all the transactions into list of lists by grouping each customers transaction's items

In [10]:

```
all_transactions = [transaction[1]['itemDescription'].tolist() for transaction in list(groceries.groupby(['Member_number', 'Date'
```

In [11]:

```
all_transactions[0:10]
```

Out[11]:

```
[['whole milk', 'pastry', 'salty snack'],
 ['sausage', 'whole milk', 'semi-finished bread', 'yogurt'],
 ['soda', 'pickled vegetables'],
 ['canned beer', 'misc. beverages'],
 ['sausage', 'hygiene articles'],
 ['sausage', 'whole milk', 'rolls/buns'],
 ['whole milk', 'soda'],
 ['frankfurter', 'soda', 'whipped/sour cream'],
 ['frankfurter', 'curd'],
 ['beef', 'white bread']]
```

Using Transaction Encoder to create a matrix with products as coloumn names

If the customer bought the product then value underlaying is True Else its False

In [13]:

```
trans_encoder = TransactionEncoder()
trans_encoder_matrix = trans_encoder.fit(all_transactions).transform(all_transactions)
trans_encoder_matrix = pd.DataFrame(trans_encoder_matrix, columns=trans_encoder.columns_)
```

We changed the data into this format as the function for calculation of Apriori rules and Fp Growth rules in mlxtend library takes this matrix as input

In [14]:

```
trans_encoder_matrix.head()
```

Out[14]:

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	bathroom cleaner	beef	berries	...	turkey	vinegar	waffles	whipped/sour cream	whisky
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
1	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
3	False	False	False	False	False	False	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

5 rows × 167 columns

Function for performing Apriori and Fp growth and finding their Execution time

In [33]:

```
def perform_rule_calculation(transact_items_matrix, rule_type="apriori", min_support=0.001):

    start_time = 0
    total_execution = 0
    start_time = time.time()
    rule_items = apriori(transact_items_matrix,
                         min_support=min_support,
                         use_colnames=True)
    total_execution = time.time() - start_time
    print("Computed Apriori!")

    rule_items['number_of_items'] = rule_items['itemsets'].apply(lambda x: len(x))

    return rule_items, total_execution
```

In [22]:

```
def perform_rule_calculation(transact_items_matrix,rule_type="fpgrowth",min_support=0.001):
    start_time=0
    total_execution=0
    start_time = time.time()
    rule_items = fpgrowth(transact_items_matrix,
                          min_support=min_support,
                          use_colnames=True)
    total_execution = time.time() - start_time
    print("Computed Fp Growth!")
    rule_items['number_of_items'] = rule_items['itemsets'].apply(lambda x: len(x))
    return rule_items,total_execution
```

Function for Computing the association rules

In [23]:

```
def compute_association_rule(rule_matrix, metric="lift", min_thresh=1):

    rules = association_rules(rule_matrix,
                              metric=metric,
                              min_threshold=min_thresh)

    return rules
```

Performing FP Growth for the Groceries dataset

In [24]:

```
fpgrowth_matrix, fp_time=perform_rule_calculation(trans_encoder_matrix)
print("Fp Growth execution took : {} seconds".format(fp_time))
```

```
Computed Fp Growth!
Fp Growth execution took : 0.5169107913970947 seconds
```

Rules generated by FP Growth

In [25]:

```
fpgrowth_matrix.head()
```

Out[25]:

	support	itemsets	number_of_items
0	0.157923	(whole milk)	1
1	0.051728	(pastry)	1
2	0.018780	(salty snack)	1
3	0.085879	(yogurt)	1
4	0.060349	(sausage)	1

In [26]:

```
fpgrowth_matrix.tail()
```

Out[26]:

	support	itemsets	number_of_items
745	0.001403	(yogurt, chewing gum)	2
746	0.001069	(other vegetables, chewing gum)	2
747	0.001002	(soda, chewing gum)	2
748	0.001069	(pasta, whole milk)	2
749	0.001002	(seasonal products, rolls/buns)	2

In [28]:

```
fp_growth_rule_lift = compute_association_rule(fpgrowth_matrix)
```

Generation of association rules

In [54]:

```
top=fp_growth_rule_lift.head()
top[['antecedents','consequents','antecedent support','consequent support','support','confidence','lift']]
```

Out[54]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(sausage)	(pastry)	0.060349	0.051728	0.003208	0.053156	1.027617
1	(pastry)	(sausage)	0.051728	0.060349	0.003208	0.062016	1.027617
2	(sausage)	(salty snack)	0.060349	0.018780	0.001136	0.018826	1.002475
3	(salty snack)	(sausage)	0.018780	0.060349	0.001136	0.060498	1.002475
4	(salty snack)	(canned beer)	0.018780	0.046916	0.001002	0.053381	1.137802

In [55]:

```
end=fp_growth_rule_lift.tail()
end[['antecedents','consequents','antecedent support','consequent support','support','confidence','lift']]
```

Out[55]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
235	(cat food)	(tropical fruit)	0.011829	0.067767	0.001002	0.084746	1.250543
236	(yogurt)	(chewing gum)	0.085879	0.012030	0.001403	0.016342	1.358508
237	(chewing gum)	(yogurt)	0.012030	0.085879	0.001403	0.116667	1.358508
238	(seasonal products)	(rolls/buns)	0.007084	0.110005	0.001002	0.141509	1.286395
239	(rolls/buns)	(seasonal products)	0.110005	0.007084	0.001002	0.009113	1.286395

In [34]:

```
fp_growth_rule = compute_association_rule(fpgrowth_matrix, metric="confidence", min_thresh=0.2)
fp_growth_rule.head()
```

Out[34]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(sausage, yogurt)	(whole milk)	0.005748	0.157923	0.001470	0.255814	1.619866	0.000563	1.131541
1	(sausage, rolls/buns)	(whole milk)	0.005347	0.157923	0.001136	0.212500	1.345594	0.000292	1.069304

Apriori Algorithm

Performing Apriori Algorithm on the groceries dataset

In [35]:

```
apriori_matrix,ap_time=perform_rule_calculation(trans_encoder_matrix, rule_type="apriori")
print("The time consumed by apriori is {}".format(ap_time))
```

```
Computed Apriori!
The time consumed by apriori is 4.332183361053467
```

Viewing the rules generated by Apriori Algorithm

In [36]:

```
apriori_matrix.head()
```

Out[36]:

	support	itemsets	number_of_items
0	0.004010	(Instant food products)	1
1	0.021386	(UHT-milk)	1
2	0.001470	(abrasive cleaner)	1
3	0.001938	(artif. sweetener)	1
4	0.008087	(baking powder)	1

In [37]:

```
apriori_matrix.tail()
```

Out[37]:

	support	itemsets	number_of_items
745	0.001136	(sausage, rolls/buns, whole milk)	3
746	0.001002	(rolls/buns, soda, whole milk)	3
747	0.001337	(rolls/buns, yogurt, whole milk)	3
748	0.001069	(sausage, soda, whole milk)	3
749	0.001470	(sausage, yogurt, whole milk)	3

In [68]:

```
apriori_rules= compute_association_rule(apriori_matrix)
top=apriori_rules.head()
top[['antecedents','consequents','antecedent support','consequent support','support','confidence','lift']]
```

Out[68]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(tropical fruit)	(UHT-milk)	0.067767	0.021386	0.001537	0.022682	1.060617
1	(UHT-milk)	(tropical fruit)	0.021386	0.067767	0.001537	0.071875	1.060617
2	(beef)	(brown bread)	0.033950	0.037626	0.001537	0.045276	1.203301
3	(brown bread)	(beef)	0.037626	0.033950	0.001537	0.040853	1.203301
4	(citrus fruit)	(beef)	0.053131	0.033950	0.001804	0.033962	1.000349

Finding the rules which satisfies for all support > 0.2

In [69]:

```
apriori_rule = compute_association_rule(apriori_matrix, metric="confidence", min_thresh=0.2)
apriori_rule.head()
```

Out[69]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(sausage, rolls/buns)	(whole milk)	0.005347	0.157923	0.001136	0.212500	1.345594	0.000292	1.069304
1	(sausage, yogurt)	(whole milk)	0.005748	0.157923	0.001470	0.255814	1.619866	0.000563	1.131541

Function for comparing the two algorithms execution time

In [70]:

```
def compare_time_exec(algo1=list, alg2=list):

    execution_times = [algo1[1], algo2[1]]
    algo_names = (algo1[0], algo2[0])
    y=np.arange(len(algo_names))

    plt.bar(y,execution_times,color=['orange', 'blue'])
    plt.xticks(y,algo_names)
    plt.xlabel('Algorithms')
    plt.ylabel('Time')
    plt.title("Execution Time (seconds) Comparison")
    plt.show()
```

In [71]:

```
algo1 = ['Fp Growth', fp_time]
algo2 = ['Apriori', ap_time]

compare_time_exec(algo1, algo2)
```