Application of Data Science techniques on a Retail Database

- by Onkar Suhas Samant

Business Questions

1) What items should we club together for giving **promotional combo offers**?
2) Which items should we place close to each other for **on-shelf / online placement**?
3) On the website, which **items should we suggest** the customers to buy?
4) What insights can we draw from the **customer purchase patterns**?
5) Which items would be ideal for implementing the **loss-leader pricing** strategy?

Loss-leader pricing strategy: https://en.wikipedia.org/wiki/Loss_leader (https://en.wikipedia.org/wiki/Loss_leader)

Step by Step Data-driven Solution

1. Data Information

Understanding data and attributes from table schema or table definition

Following open-source dataset has been used: https://archive.ics.uci.edu/ml/datasets/Online+Retail (https://archive.ics.uci.edu/ml/datasets/Online+Retail)
The dataset contains transaction data of an online retailer based in the UK for a year.
It is made publicly available by the School of Engineering, London South Bank University. It has **541,909 records**. It has the following attributes:

1) **InvoiceNo** : 6 digit transaction ID
2) **StockCode**: Product ID
3) **Description**: Product Description
4) **Quantity**: Counts of the product purchased
5) **InvoiceDate**: Date and time of transaction
6) **UnitPrice** Price of 1 item
7) **CustumerID**: Unique ID for a customer
8) **Country**: Country where the customer resides

2. Data Science Project Goal

Defining goals from data science perspective to solve the business questions

1) Identify frequently bought items i.e. **frequent item-sets** in transaction DB for:
   a) Purchase Behaviour Analysis: To get more Insights on customer purchase behavior
   b) Product Placement: Identifying products that may often be purchased together and arranging the placement of those close by to encourage the purchaser to buy both items. This placement can be physical (on-shelf) or virtual (on an e-commerce site).
   c) Items Suggestions: for the customer based on the purchase of some other items
2) Build **association rules** from the identified frequent item-sets for:
   a) Promotional Offers: If product A is often bought with product B, both products can be combined for an ideal promotional combo offer
   b) Loss leader pricing: Often a retailer sells one product at a loss which drives sales for another item (from which the profit is recovered). A simple example: A loss on shampoo sale can be recovered from its complementary items like conditioner and hair serum

3. Algorithms

Selecting the most appropirate algorithms for the tasks

Frequent item-set mining using following algorithms:
1) Apriori
2) FP-Growth
Association Rule Mining on the frequent item-sets

Links for studying both the algorithms:
Apriori: https://en.wikipedia.org/wiki/Apriori_algorithm (https://en.wikipedia.org/wiki/Apriori_algorithm)
FPGrowth: https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm (https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm)

Reason for selecting these algorithms:
1) Both algorithms are more efficient than *Brute force counting* of frequent itemsets in terms of time complexity and thus can be scaled for a large retail transaction DB
2) In both Apriori and FP-Growth, candidate item-set generation is optimal.

4. A Deep Dive Into Data

Any Data Science work is as good as it's data. Let's get an **in-depth understanding of our data** before we proceed with the imlemetation of algorithms for our tasks

In [1]:
```
#Importing useful libraries for data analysis using data frames and visualization libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#importing data and priting first few transactions
online_retail  = pd.read_excel("Online Retail.xlsx")
online_retail.head()
```

Out[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

4a) Analysis of the raw data and basic data cleaning

In [3]:
```
print("Number of points:",online_retail.shape[0])
print("Number of dimensions:",online_retail.shape[1])
```

```
Number of points: 541909
Number of dimensions: 8
```

In [4]:
```
#We create a copy of our original data for any modidcations (important ot have the original data seperately)
retail_df = online_retail.copy()
#deleting junk records with qunatities 0 or less than 0 (probably returned items)
retail_df["Description"] = retail_df["Description"].astype(str)
retail_df["StockCode"] = retail_df["StockCode"].astype(str)
retail_df = retail_df[retail_df["StockCode"].str.contains('[0-9]')]
retail_df= retail_df[retail_df['UnitPrice']!=0]
retail_df= retail_df[retail_df['Quantity']>0]
#deleting cancelled records
retail_df['InvoiceNo'] = retail_df['InvoiceNo'].astype(str)
retail_df = retail_df[~retail_df['InvoiceNo'].str.contains('C')]
#checking for null values in columns
print("Total null values in every column")
print(retail_df.isna().sum())
```

```
Total null values in every column
InvoiceNo           0
StockCode           0
Description         0
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     131436
Country             0
dtype: int64
```

We see there are no customer IDs for a lot of records. These records would most probably be of customers who did not use their identification card while purchasing. However, we can continue to use these records for itemset analysis, but we have to be aware of this if we are to do any customer segmentation exercise.

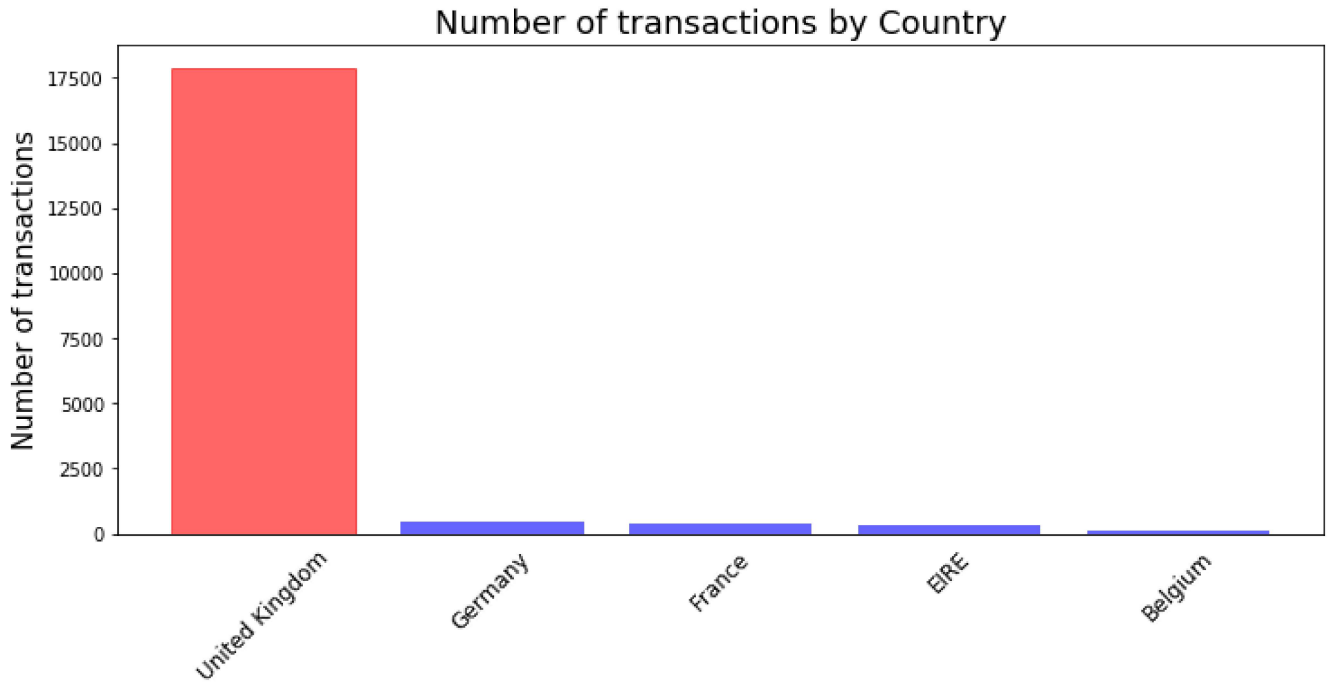
In [5]:
```
#Check for unique entries in each column
retail_df.nunique()
```

Out[5]:
```
InvoiceNo      19776
StockCode       3911
Description     4016
Quantity         374
InvoiceDate    18335
UnitPrice        502
CustomerID      4334
Country           38
dtype: int64
```

We observe the StockCode (Product ID) to Description mapping is not one to one. This is a data quality issue which we have to take into account

4b) Visualizing the data

In [6]:
```
trans_by_country = retail_df.groupby("Country")["InvoiceNo"].nunique()
trans_by_country5 = trans_by_country.nlargest(5)
plt.figure(figsize=(12,5))
plt.title('Number of transactions by Country', fontsize = 18)
plt.ylabel('Number of transactions', fontsize=15)
plt.xlabel('')
plt.tick_params(axis='x', which='both', bottom=False,top=False)
#plt.xlabel('Country',fontweight='bold', fontsize=12)
barlist = plt.bar(trans_by_country5.index, trans_by_country5.values, alpha = 0.6, color = "b")
barlist[0].set_color('r')
x = plt.gca().xaxis
#plt.legend(loc=4, frameon=False, title='Legend')
# rotate the tick labels for the x axis
for item in x.get_ticklabels():
    item.set_fontsize(12)
    item.set_rotation(45)
```

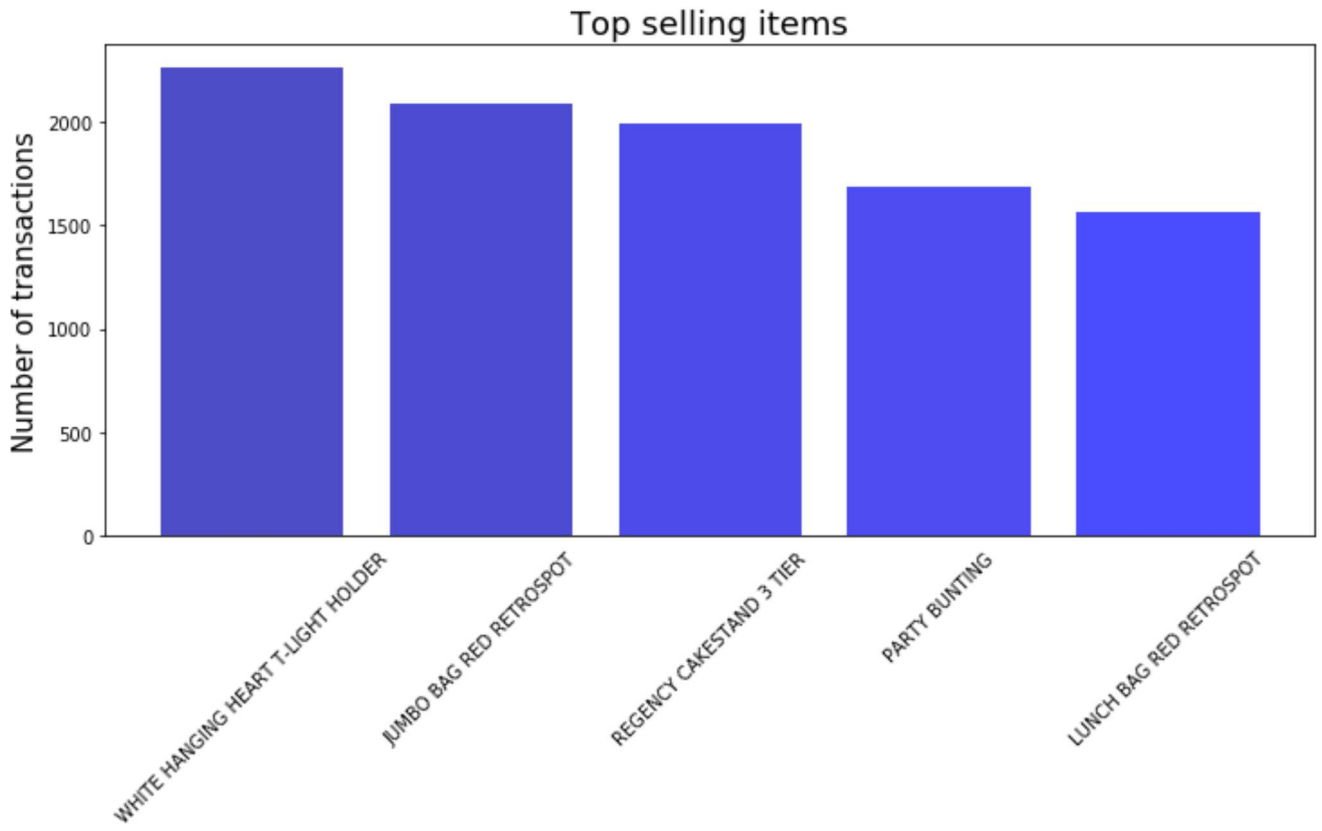


In [7]:
```
#Percentage of total transaction by customers of UK
round(trans_by_country5[0] / sum(trans_by_country) * 100,2)
```

Out[7]: 90.52

We observe this retailer is UK based hence **most customers are from the UK**, which is around **90.52%** of the total customers

In [8]:
```
trans_by_product = retail_df.groupby("Description")["InvoiceNo"].nunique()
trans_by_product5 = trans_by_product.nlargest(5)
#trans_by_country[trans_by_country > 100].plot(kind="bar")
plt.figure(figsize=(12,5))
plt.title('Top selling items', fontsize = 18)
plt.ylabel('Number of transactions', fontsize=15)
plt.xlabel('')
colors_rgb = np.zeros((5,4))
blues = trans_by_product5.values / trans_by_product5.values[0]
colors_rgb[:,2] = blues[::-1] #np.linspace(0.6,1,5)
#colors_rgb[:,1] = np.linspace(1,0,5)
plt.bar(trans_by_product5.index, trans_by_product5.values, alpha = 0.7, color=colors_rgb)
plt.tick_params(axis='x', which='both', bottom=False,top=False)
x = plt.gca().xaxis
for item in x.get_ticklabels():
    item.set_rotation(45)
plt.show()
```



Percentage of total transaction by the top 5 highest selling items

In [9]:
```
print("Total number of items = ",trans_by_product.count())
print("Percentage contribution of the top 5 selling items to the total transactions = "+str(roun
d((sum(trans_by_product5) / sum(trans_by_product)) * 100,2))+"%")
```

```
Total number of items =  4016
Percentage contribution of the top 5 selling items to the total transactions = 1.85%
```

We observe that the contribution of the top 5 high selling items is not very significant. However, it gives us a good insight into which items are the main drivers of total sale.

5. Applying Algorithms

Support: a metric which indicates how frequent is the item / item-set in the transaction DB

In [10]:
```
#Loading the libraries
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.preprocessing import TransactionEncoder
```

In [11]:
```
#Changing format of the dataset to pass to the library Apriori and FPGrowth functions
def format_data(original_df):
    trans_df = original_df.groupby('InvoiceNo')['Description'].apply(list)
    trans_list = trans_df.to_list()
    te = TransactionEncoder()
    te_ary = te.fit(trans_list).transform(trans_list)
    #print(trans_list[0:10])
    return pd.DataFrame(te_ary, columns=te.columns_)
```

In [12]:
```
trans_items_df = format_data(retail_df)
```

In [13]:
```
#Now we have the data in the format we want, True values refer to the item has been purchased in that transaction
trans_items_df.head()
```

Out[13]:

	4 PURPLE FLOCK DINNER CANDLES	50'S CHRISTMAS GIFT BAG LARGE	DOLLY GIRL BEAKER	I LOVE LONDON MINI BACKPACK	I LOVE LONDON MINI RUCKSACK	NINE DRAWER OFFICE TIDY	OVAL WALL MIRROR DIAMANTE	RED SPOT GIFT BAG LARGE	SET 2 TEA TOWELS I LOVE LONDON	SPACEBOY BABY GIFT SET
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False

5 rows × 4016 columns

We need to select the following before we find the frequent item-sets:

a) **Algorithm**: Apriori vs FPGrowth: We can compare the performances of both the algorithms on a smaller subset of data

b) **min_support**: We need min_support value ideal for our goal mentioned above

5a) Perfomance comparison of Apriori vs FPGrowth

In [14]:
```
#Create a smaller copy of the original data frame for this and format it into transaction vs items foramt
trans_items_df2 = trans_items_df.iloc[0:2000, 0:400]
```

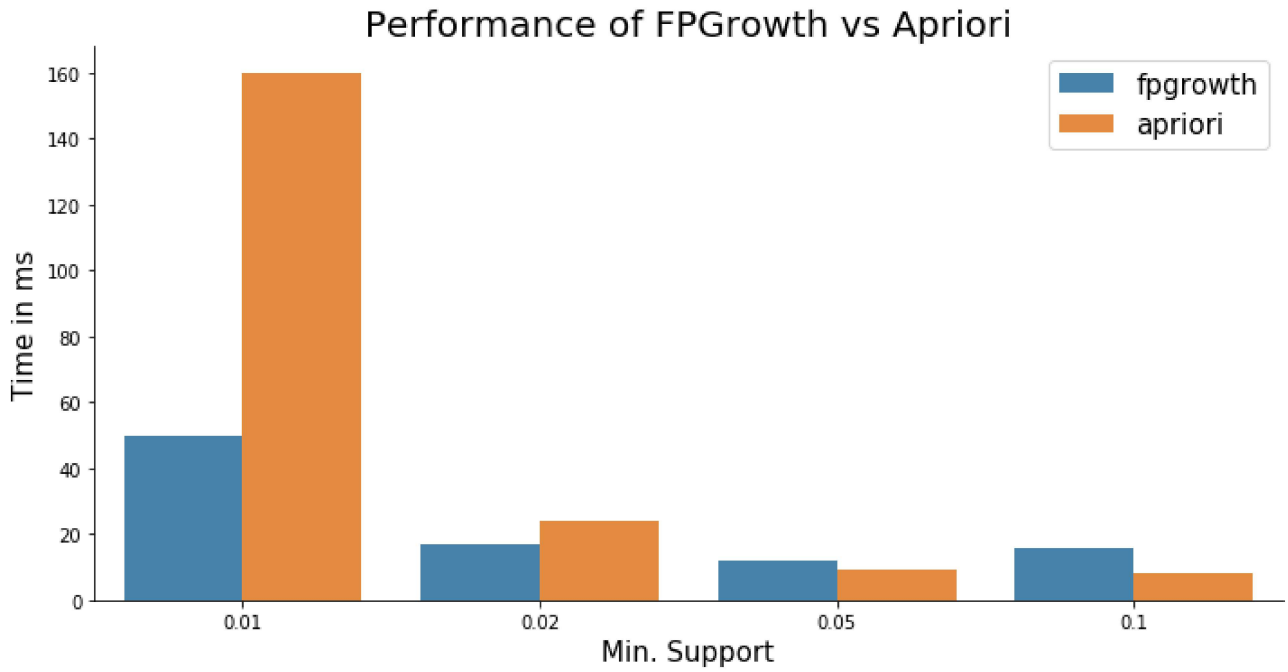
In [15]:
```
#We check perfomance for different hyperparameters i.e. diff min_support values
min_support = [0.1,0.05,0.02, 0.01]
time_comp = pd.DataFrame(columns = ["algo", "min_support", "time(ms)"])
```

In [16]:
```
import time
for ms in min_support:
    tick = time.time()
    fpgrowth(trans_items_df2, min_support = ms, use_colnames=True)
    tock = time.time()
    time_comp = time_comp.append({"algo":"fpgrowth", "min_support":ms, "time(ms)":round((tock - tick)*1000,0)},ignore_index=True)

    tick = time.time()
    apriori(trans_items_df2, min_support = ms, use_colnames=True)
    tock = time.time()
    time_comp = time_comp.append({"algo":"apriori", "min_support":ms, "time(ms)":round((tock - tick)*1000,0)},ignore_index=True)
```

In [17]:
```
sns.catplot(x = 'min_support', y='time(ms)', hue = 'algo',data=time_comp, kind='bar', legend = F
alse, alpha = 0.9, height=5, aspect=2)
plt.legend(prop={'size': 15})
plt.title('Performance of FPGrowth vs Apriori', fontsize = 20)
plt.ylabel('Time in ms', fontsize=15)
plt.xlabel('Min. Support',fontsize=15 )
```

Out[17]:
```
Text(0.5, 6.79999999999999, 'Min. Support')
```



We observe ***FPGrowth performs a lot better*** on smaller min_support value and is comparable for larger values
This is due to the fact FPGrowth refers to the original transaction DB only once and then creates suffix tree for candidate generation unlike Apriori algorithm where transaction DB is referred at every stage of item-set generation
Therefore, **we can select FPGrowth for the larger dataset**

5b) Selecting the apriopriate min_support value

We want a min_support value such that:

1) We are able to select item-sets of size more than 1
2) It does not take a lot of time to run. Smaller the min_support value more the time it will take to run the algorithm

In [18]:
```
#Create a dictionary for storing frequent itemsets
#freq_is1 = frequent item sets of size 1
#freq_is2 = frequent item sets of size 2 or more
freq_is = {0.1:0,0.05:0, 0.02:0, 0.01:0}
freq_is1 = {0.1:0,0.05:0, 0.02:0, 0.01:0}
freq_is2 = {0.1:0,0.05:0, 0.02:0, 0.01:0}
```

In [19]:
```
for ms in min_support:
    frequent_itemsets = fpgrowth(trans_items_df, min_support=ms, use_colnames=True)
    frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
    print("min support = "+str(ms))
    freq_is[ms] = len(frequent_itemsets)
    freq_is1[ms] = sum(frequent_itemsets["length"]==1)
    freq_is2[ms] = sum(frequent_itemsets["length"]>1)
    print("total frequent itemsets of size 1 = "+str(freq_is1[ms]))
    print("total frequent itemsets of size more than 1 = "+str(freq_is2[ms]))
```

```
min support = 0.1
total frequent itemsets of size 1 = 3
total frequent itemsets of size more than 1 = 0
min support = 0.05
total frequent itemsets of size 1 = 33
total frequent itemsets of size more than 1 = 0
min support = 0.02
total frequent itemsets of size 1 = 299
total frequent itemsets of size more than 1 = 82
min support = 0.01
total frequent itemsets of size 1 = 823
total frequent itemsets of size more than 1 = 1051
```

We observe:

1) **min_support = 0.02** should be okay for creating association rules from the frequent item-sets as it has enough frequent item-sets of size more than 1

2) The number of frequent item-sets more than 1 is not very large so it will create association rules faster than for item-sets of min_support = 0.01

6. Generating frequent item-sets with:

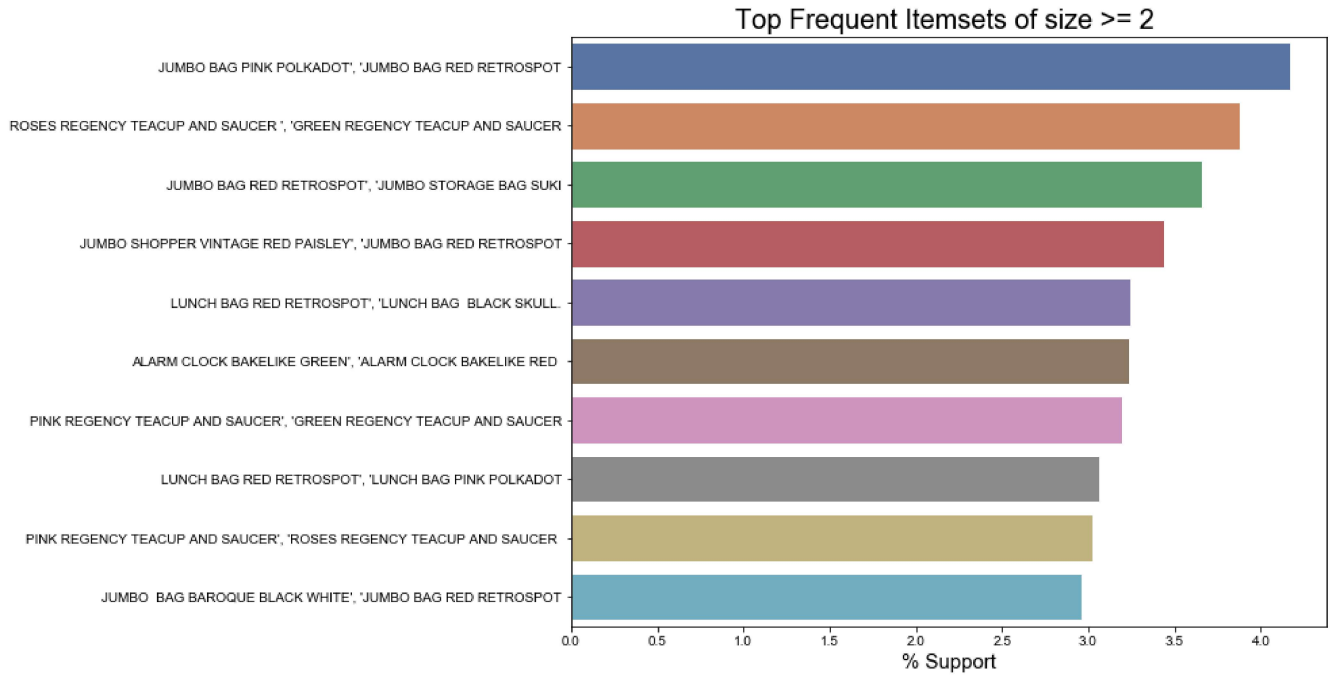
a) FPGrowth algorithm

b) min_support = 0.02 (2%)

In [20]:
```
frequent_itemsets = fpgrowth(trans_items_df, min_support=0.02, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets2 = frequent_itemsets[frequent_itemsets["length"]>=2].sort_values(by=['support'
], ascending=False)
Top10_s = list(frequent_itemsets2["support"][0:10]*100)
Top10_is = list(frequent_itemsets2["itemsets"][0:10])
X = [list(x) for x in Top10_is]
Top10_is = [str(x)[2:-2] for x in X]
```

In [21]:
```
plt.figure(figsize=(10,8))
plt.title('Top Frequent Itemsets of size >= 2 ', fontsize = 20)
plt.xlabel('% Support', fontsize=15)
sns.set(font_scale=0.8)
sns.barplot(Top10_s,Top10_is)
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1904728ac18>
```



Following insights can be drawn from this analysis

1) **Jumbo Bag Red Retrospot** is one of the highest selling item (from high selling item chart) and also forms frequent itemsets with others. Such items can be sold on discount to drive sales of other items bought frequently with it like : Jumbo Storage Bad Suki , Jumbo Shopper Vintage Red Paisley

2) Purcahse beahviour patterns: Customers buy **same types of items together**: eg: Jumbo bags of 2 types or Alarm clocks of 2 types or 2 types of teacups and saucer

3) Online product placement: On the website **these combinations can be put closer to eachother** for driving more sales of items which have high affinity with the other

7. Building Association Rules

We try 2 important criterias:

1) **Cofidence**: a metric which indicates how often is the rule found to be true

confidence (X → Y) = sup(XY) / sup(X)

In [22]:
```
from mlxtend.frequent_patterns import association_rules
rules_c = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
```

In [23]:
```
rules_c[0:1]
```

Out[23]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(JUMBO BAG PINK POLKADOT, JUMBO STORAGE BAG SUKI)	(JUMBO BAG RED RETROSPOT)	0.026042	0.105633	0.020884	0.801942	7.591766	0.018133	4.515676

Interpretation of Confidence:
Customers buying JUMBO BAG RED RETROSPOT item buy item-set (JUMBO BAG PINK POLKADOT, JUMBO STORAGE BAG SUKI) along with it 80% of the time
Therefore the rule, **(JUMBO BAG RED RETROSPOT) → (JUMBO BAG PINK POLKADOT, JUMBO STORAGE BAG SUKI)** is quite a strong rule

2) **Lift**: a metric which determines the stregth of the rule over random occurence
lift(X → Y) = sup(XY) / (sup(X) * sup(Y))

In [24]:
```
rules_l = association_rules(frequent_itemsets, metric = "lift", min_threshold  = 1.2)
```

In [25]:
```
rules_l[0:1]
```

Out[25]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(JAM MAKING SET WITH JARS)	(JAM MAKING SET PRINTED)	0.057241	0.058758	0.023867	0.416961	7.096233	0.020504	1.614372

Interpretation of Lift:
Notion of how independent are the antecedent and consequent item-sets from eachother.
a) lift >> 1: Highly dependent on eachother
b) lift = 1: Independent of eachother
c) lift << 1: Items are substitutes of each other (if of the same class eg: Pepsi and Coke belonging to the Beverages class etc.)

There are other important criterias like leverage and conviction

7a) Building some important association rules for practical applications

We can select the rules having **high confidence and lift** i.e.
1) **confidence >= 0.5**
2) **lift >= 2**

In [26]:
```
rules = association_rules(frequent_itemsets, metric = "confidence", min_threshold = 0.6)
rules_imp = rules[(rules["lift"] > 2)].copy()
```

7b) Insights from these rules

In [27]:
```
rules_imp.head(1)
```

Out[27]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.049555	0.053145	0.032362	0.653061	12.288239	0.029729	2.72917

(1) **Promotional Offers and Item Suggestions:**

a) A **Promotional Combo Offer** can be created for buying the items (ALARM CLOCK BAKELIKE RED, ALARM CLOCK BAKELIKE GREEN) together.

The rule is very strong and the customers buying one item are likely to buy the second item too

b) **Item Suggestions**: A customer buying the item (or having in the cart) ALARM CLOCK BAKELIKE GREEN should be suggested the ALARM CLOCK BAKELIKE RED on the website as it's highly likely the customer will buy both the items

(2) **Loss Leader Pricing:** To implement this we need antecedents to be more than 1 so as to recover loss from multiple other items

In [29]:
```
rules_imp_ll = rules_imp.copy()
rules_imp_ll["antecedent_len"] = rules["antecedents"].apply(lambda x: len(x))
rules_imp_ll = rules_imp_ll[rules_imp_ll["antecedent_len"]>=2]
```

In [30]:
```
rules_imp_ll.head(1)
```

Out[30]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecede
6	(JUMBO BAG PINK POLKADOT, JUMBO STORAGE BAG SUKI)	(JUMBO BAG RED RETROSPOT)	0.026042	0.105633	0.020884	0.801942	7.591766	0.018133	4.515676	

This is an imporant association rule set for applying loss leader pricing strategy in retail. An example of the implementation from the above set is:
JUMBO BAG RED RETROSPOT (consequent) can be sold at a discount, as the higher sale of this item due to a discount will in turn **increase the sale of JUMBO BAG PINK POLKADOT and JUMBO STORAGE BAG SUKI** (antecedents)

This exercise gives an important application of Data Science techniques for solving some of the most common business questions in retail. The examples mentioned are limited to a few for simplicity