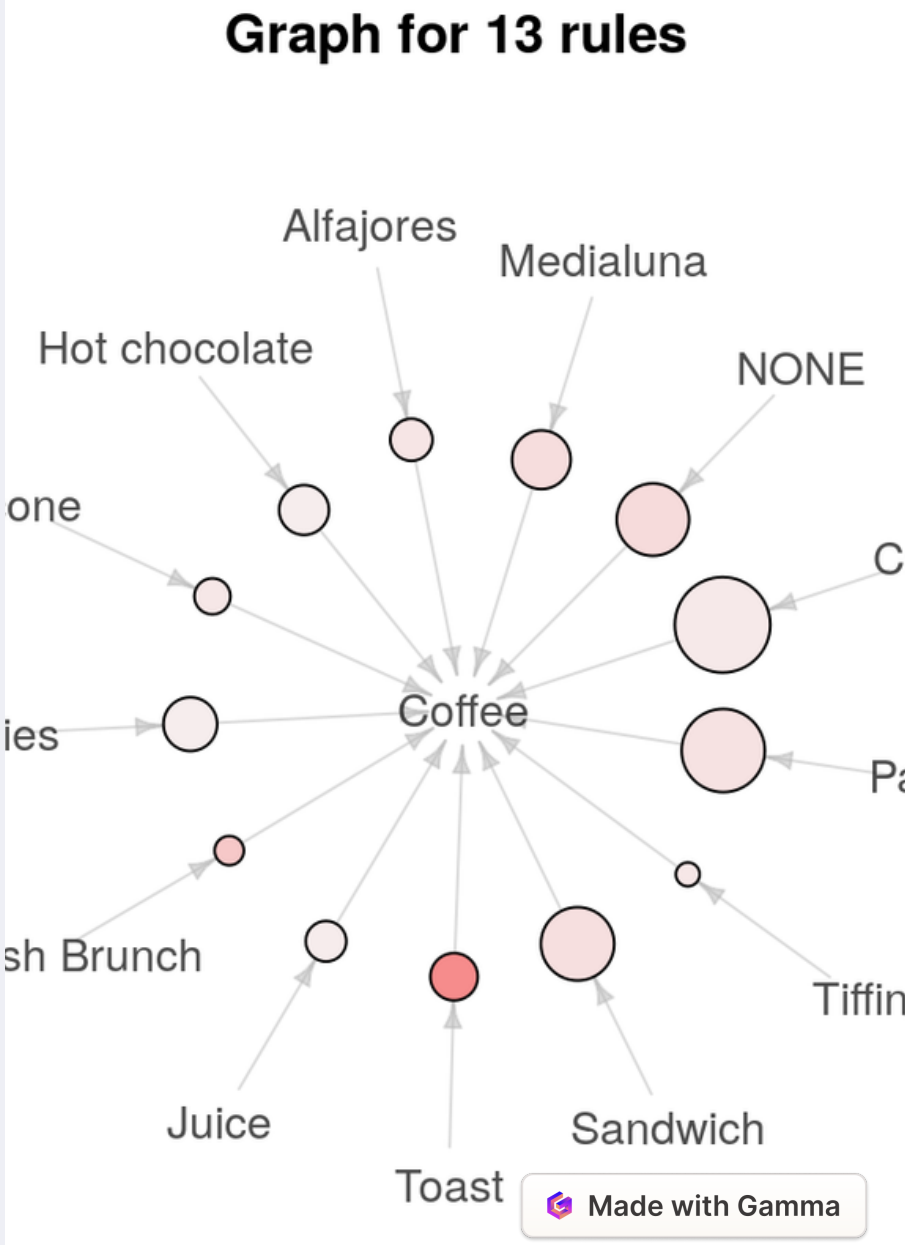
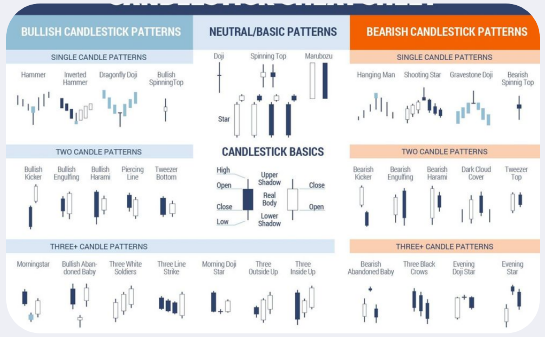
Market Basket Analysis

Analyzing customer purchase patterns to uncover insights and optimize product placement and promotions.

by Ajay Kumar

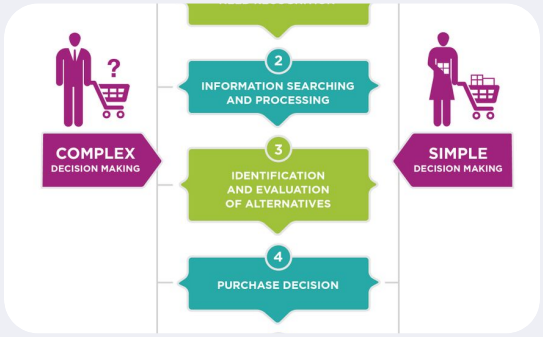


Benefits of Market Basket Analysis



Insightful Patterns

Reveals correlations between product purchases



Customer Understanding

Enhances understanding of customer preferences

Market Trends

Identifies emerging market trends and consumer behavior

Key Insights from Market Basket Data

Association Rules

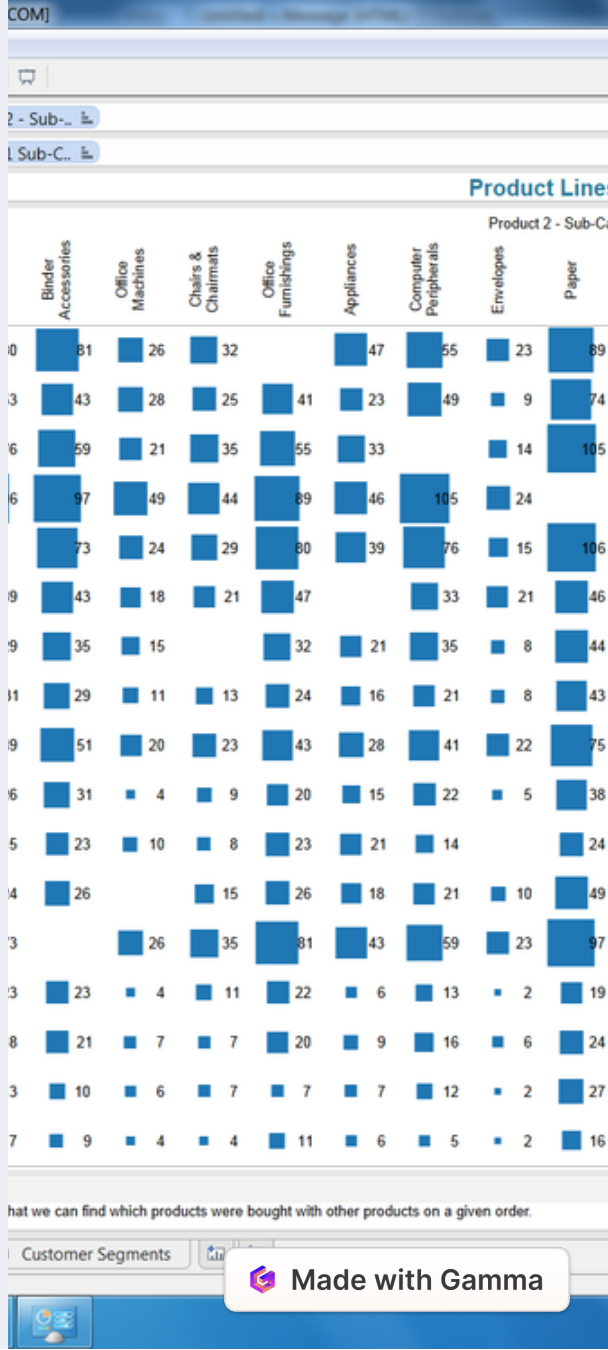
Identify rules for product combinations and recommendations

Customer Segmentation

Cluster customers based on similar purchasing behavior

Cross-Selling Opportunities

Recommending related products to increase sales



Market Basket Analysis: Process Overview

1

Data Collection

Collect transactional data from various sources

2

Preprocessing

Clean and prepare the data for analysis

3

Association Mining

Use algorithms to discover patterns and associations

4

Insights Generation

Generate actionable insights for business decisions

Challenges in Market Basket Analysis

1 **Data Complexity**

Dealing with large volumes of transactional data

2 **Algorithm Selection**

Choosing appropriate algorithms for accurate results

3 **Interpreting Results**

Understanding and properly interpreting the discovered patterns

4 **Privacy Concerns**

Ensuring customer privacy and compliance with regulations

Application of Market Basket Analysis

Retail

Optimizing product placement, promotions, and stock management

Improving customer experience with personalized recommendations

E-commerce

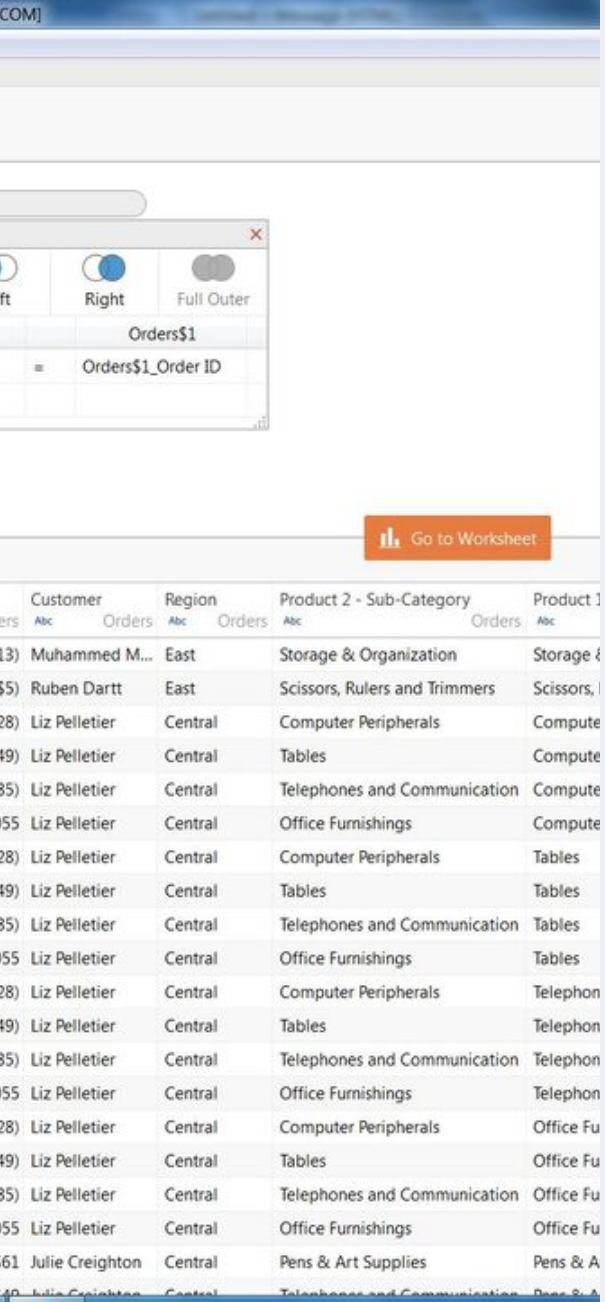
Enhancing cross-selling and upselling strategies

Developing targeted marketing campaigns based on customer behavior

Grocery Stores

Creating effective bundling of products and discount offers

Identifying popular product combinations for strategic pricing



Future Trends in Market Basket Analysis

1

AI and Machine Learning

Advanced algorithms for more accurate and real-time insights

2

Big Data Integration

Utilizing vast amounts of data from multiple sources for comprehensive analysis

3

Predictive Analytics

Anticipating future purchases and consumer behavior for proactive strategies

Implementing Market Basket Analysis

1. Identify Business Objectives	Understand the specific goals and objectives for analysis
2. Data Collection and Preparation	Gather and clean relevant transactional data for analysis
3. Analysis and Insights Generation	Apply algorithms to discover patterns and generate actionable insights
4. Implementation and Monitoring	Implement insights into strategies and monitor impact on business outcomes

Market Basket Analysis (MBA):-

There are two parts in this project:

1. Understanding MBA
2. Implementation in Python '''

Understanding MBA:-

Market basket analysis (MBA), also known as Association-Rule Mining, is a useful method of discovering customer purchasing patterns by extracting associations or co-occurrences from stores' transactional databases.

Example:-

1)-If you are in a supermarket and you buy a loaf of Bread, you are more likely to buy a packet of Butter at the same time than somebody who didn't buy the Bread.

2)-If you are buying a XiaoMi Power Bank in an online store, you are more likely to also buy a carrying case to go with the power bank. Amazon knows this well from the transaction data of its millions of customers.

Application:-

1)-Recommendation Engine:-

Showing related products as "Customers Who Bought This Item Also Bought" or “Frequently bought together” (as shown in the Amazon example above). It can also be applied to recommend videos and news article by analyzing the videos or news articles that are often watched or read together in a user session.

2)-Cross-sell/bundle products:-

Selling associated products as a "bundle" instead of individual items. For example, transaction data may show that customers often buy a new phone with screen protector together. Phone retailers can then package new phone with high-margin screen protector together and sell them as a bundle, thereby increasing their sales.

3)-Arrangement of items in retail stores-

Associated items can be placed closer to each other, thereby invoking "impulse buying". For example it may be uncovered that customers who buy Barbie dolls also buy candy at the same time. Thus retailers can place high-margin candy near Barbie doll display, thereby tempting customers to buy them together.

4)-Detecting fraud-

Identifying related actions whenever a fraudulent transaction is performed. For example, in a fraudulent insurance claim for stolen vehicle, it may be analyzed (from historical data) that claimant frequently report the incident a few days late (action 1) and often refuse to cooperate with insurer on investigation (action 2). Insurers can identify these red flags once certain behaviors or actions are displayed by the claimants.'''

Case Study:-

For simplicity we are analyzing only 2 items-Bread and Butter. We want to know if there is any evidence that suggests that buying Bread leads to buying Butter.

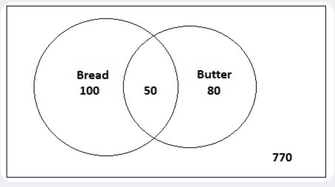
Problem Statement: Is the purchase of Bread leads to the purchase of Butter?

Hypothesis: There is significant evidence to show that buying Bread leads to buying Butter.

Bread => Butter (Buying together)

Antecedent => Consequent

Let's take the example of a supermarket which generates 1,000 transactions monthly, of which Bread was purchased in 150 transactions, Butter in 130 transactions, and both together in 50 transactions, as shown in the Venn diagram below:



Analysis and Findings:-

We can use MBA to extract the association rule between Bread and Butter. There are three metrics or criteria to evaluate the strength or quality of an association rule, which are support, confidence and lift.

1)-Support:-

Support measures the percentage of transactions containing a particular combination of items relative to the total number of transactions. In our example, this is the percentage of transactions where both Bread and Butter are bought together. We need to calculate this to know if this combination of items is significant or negligible? Generally, we want a high percentage i.e. high support in order to make sure it is a useful relationship. Typically, we will set a threshold,

for example we will only look at a combination if more than 1% of transactions have this combination.

Support (antecedent (Bread) and consequent (Butter)) = Number of transactions having both items / Total transactions P(Bread ∩ Butter) Number of Transection With Butter And Bread/Total Number of Transection

$$P(\text{Bread INTERSECTION Butter})$$

$$= P(\text{Bread} \frown \text{Butter})$$

$$= \frac{\text{Number of transactions with Bread AND Butter}}{\text{Total transactions}}$$

$$= \frac{50}{1000}$$

$$= 5\%$$

Result:-

The support value of 5% means 5% of all transactions have this combination of Bread and Butter bought together. Since the value is above the threshold of 1%, it shows there is indeed support for this association and thus satisfy the first criteria.

2. Confidence

Confidence measures the **probability of finding a particular combination of items whenever antecedent is bought**. In probability terms, **confidence is the conditional probability of the consequent given the antecedent** and is represented as **P (consequent / antecedent)**. In our example, it is the probability of both Bread and Butter being bought together whenever Bread is bought. Typically, we may set a threshold, say we want this combination to occur at least **25%** of times when Bread is bought.

Confidence (antecedent i.e. Bread and consequent i.e. Butter) = P (Consequent (Butter) is bought GIVEN antecedent (Bread) is bought)

$$P(\text{Butter GIVEN Bread}) = \frac{P(\text{Bread} \cap \text{Butter})}{P(\text{Bread})}$$

$$= \frac{\text{Number of transactions with Bread AND Butter}}{\text{Number of transactions with Bread}}$$

$$= \frac{50}{150}$$

$$= 33.3\%$$

Result: The confidence value of 33.3% is above the threshold of 25%, indicating we can be ***confident*** that Butter will be bought whenever Bread is bought, and thus satisfy the second criteria.

3. Lift

Lift is a metric to determine how much the purchase of antecedent influences the purchase of consequent. In our example, we want to know **whether the purchase of Butter is independent of the purchase of Bread (or**) is the **purchase of Butter happening due to the purchase of Bread?** In probability terms, we want to know which is higher, P (Butter) or P (Butter / Bread)? If the purchase of Butter is influenced by the purchase of Bread, then P (Butter / Bread) will be higher than P (Butter), or in other words, the ratio of P (Butter / Bread) over P (Butter) will be higher than 1.

Formula,

$$= \frac{P(\text{Bread} \cap \text{Butter})}{P(\text{Bread})} > P(\text{Butter})$$

$$= \frac{P(\text{Bread} \cap \text{Butter})}{P(\text{Bread}) * P(\text{Butter})} > 1$$

$$= \frac{\frac{50}{100}}{\frac{150}{1000} \times \frac{130}{1000}}$$

$$= 2.56$$

Intuitively,

$$= \frac{P(\text{Bread} \cap \text{Butter})}{P(\text{Bread})} > P(\text{Butter})$$

$$= \frac{50}{150} > \frac{130}{1000}$$

$$= 0.33 > 0.13$$

$$= \frac{0.33}{0.13}$$

$$= 2.56$$

Result: The lift value of 2.56 is greater than 1, it shows that the purchase of Butter is indeed influenced by the purchase of Bread rather than Butter's purchase being independent of Bread. The lift value of 2.56 also means that Bread's purchase Lifts the Butter's purchase by 2.56 times.

Conclusion

Based on the findings above, we can justify our initial hypothesis as we

A) Have the support of 5% transactions for Bread and Butter in the same basket
B) Have 33.3% confidence that Butter sales happen whenever Bread is purchased.
C) Knows the lift in Butter's sales is 2.56 times more, whenever Bread is purchased than when Butter is purchased alone.

Therefore, we can conclude that there is indeed evidence to suggest that the purchase of Bread leads to the purchase of Butter. This is a valuable insight to guide management's decision-making. For example, managers of retail stores could start placing bread and butter close to each other, knowing that customers are highly likely to "impulsively" purchase them together, thereby increasing the store's revenue.

Load data:-

In [2]:

```
# load the data into a pandas dataframe and take a look at the first 10 rows
bread = pd.read_csv("BreadBasket_DMS.csv")
bread.head(10)
```

Out[2]:

	Date	Time	Transaction	Item
0	2016-10-30	09:58:11	1	Bread
1	2016-10-30	10:05:34	2	Scandinavian
2	2016-10-30	10:05:34	2	Scandinavian
3	2016-10-30	10:07:57	3	Hot chocolate
4	2016-10-30	10:07:57	3	Jam
5	2016-10-30	10:07:57	3	Cookies
6	2016-10-30	10:08:41	4	Muffin
7	2016-10-30	10:13:03	5	Coffee
8	2016-10-30	10:13:03	5	Pastry
9	2016-10-30	10:13:03	5	Bread

Note:-

There are 21,293 rows and 4 columns in the data frame. Date and Time columns are encoded in 'object' instead of Datetime, but fortunately there is a Transaction column which helps to identify each transaction. Item column contains the individual items in that transaction.

For example, Transaction No. 3 contains items of "Hot chocolate", "Jam", and "Cookies" which are all transacted in the same time i.e. 10:07:57 on 2016-10-30.

Check for Missing Values

In [4]:
```
# check for missing values
bread.isnull().sum()
```

Out[4]:
```
Date           0
Time           0
Transaction    0
Item           0
dtype: int64
```

In [5]:
```
missing_value = ["NaN", "NONE", "None", "Nil", "nan", "none", "nil", 0]
print("There are {0} missing values in the dataframe.".format(len(bread[bread.Item.isin(missing_value)])))
bread[bread.Item.isin(missing_value)].head(10)
```

```
There are 786 missing values in the dataframe.
```

Out[5]:

	Date	Time	Transaction	Item
26	2016-10-30	10:27:21	11	NONE
38	2016-10-30	10:34:36	15	NONE
39	2016-10-30	10:34:36	15	NONE
66	2016-10-30	11:05:30	29	NONE
80	2016-10-30	11:37:10	37	NONE
85	2016-10-30	11:55:51	40	NONE
126	2016-10-30	13:02:04	59	NONE
140	2016-10-30	13:37:25	65	NONE
149	2016-10-30	13:46:48	67	NONE
167	2016-10-30	14:32:26	75	NONE

Note: While there is no empty cell in the dataframe, a check using the popular missing value shows that there are 786 rows with "NONE" in the column `Item`. Since the items are not recorded, we will have to remove these rows.

Note: -

While there is no empty cell in the data frame, a check using the popular missing value shows that there are 786 rows with "NONE" in the column Item. Since the items are not recorded, we will have to remove these rows.

In [6]:

```
bread = bread.drop(bread[bread.Item == "NONE"].index)
print("Number of rows: {0}".format(len(bread)))
bread.head(10)
```

```
Number of rows: 20507
```

Out[6]:

	Date	Time	Transaction	Item
0	2016-10-30	09:58:11	1	Bread
1	2016-10-30	10:05:34	2	Scandinavian
2	2016-10-30	10:05:34	2	Scandinavian
3	2016-10-30	10:07:57	3	Hot chocolate
4	2016-10-30	10:07:57	3	Jam
5	2016-10-30	10:07:57	3	Cookies
6	2016-10-30	10:08:41	4	Muffin
7	2016-10-30	10:13:03	5	Coffee
8	2016-10-30	10:13:03	5	Pastry
9	2016-10-30	10:13:03	5	Bread

Note: After removing the missing values, the number of rows left is 20,507 (original 21,293 minus 786 missing)

Convert to DatetimeIndex

In [7]:
```
bread['Datetime'] = pd.to_datetime(bread['Date']+' '+bread['Time'])
bread = bread[["Datetime", "Transaction", "Item"]].set_index("Datetime")
bread.head(10)
```

Out[7]:

Datetime	Transaction	Item
2016-10-30 09:58:11	1	Bread
2016-10-30 10:05:34	2	Scandinavian
2016-10-30 10:05:34	2	Scandinavian
2016-10-30 10:07:57	3	Hot chocolate
2016-10-30 10:07:57	3	Jam
2016-10-30 10:07:57	3	Cookies
2016-10-30 10:08:41	4	Muffin
2016-10-30 10:13:03	5	Coffee
2016-10-30 10:13:03	5	Pastry
2016-10-30 10:13:03	5	Bread

Quick Stats

In [8]:

```
total_items = len(bread)
total_days = len(np.unique(bread.index.date))
total_months = len(np.unique(bread.index.month))
average_items = total_items / total_days
unique_items = bread.Item.unique().size

print("There are {} unique items sold by the Bakery".format(unique_items))
print("Total {} items sold in {} days throughout {} months".format(total_items, total_days, total_months))
print("With an average of {} items sold daily".format(average_items))
```

```
There are 94 unique items sold by the Bakery
Total 20507 items sold in 159 days throughout 7 months
With an average of 128.9748427672956 items sold daily
```

Note: We have combined the `Date` and `Time` columns into a single `Datetime` column, convert it into datetime64 type, and then set it as DatetimeIndex. This will make it easier to plot the time series charts later on. Also, a quick look at the data shows that the Bakery sold an average of 129 items daily.

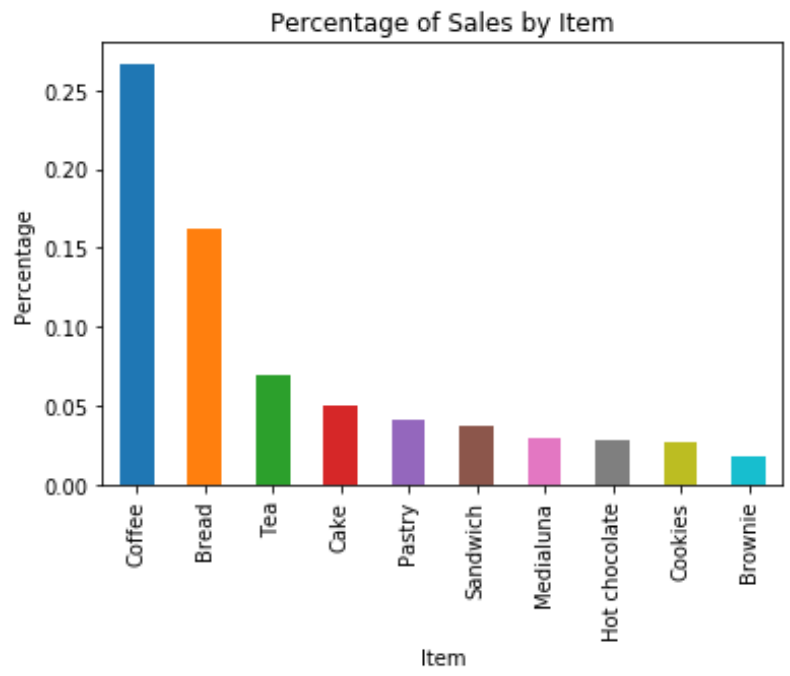
Visualization

In [9]:
```
# rank the top 10 best-selling items
bread.Item.value_counts(normalize=True)[:10]
```

Out[9]:
```
Coffee          0.266787
Bread           0.162140
Tea             0.069976
Cake            0.049983
Pastry          0.041742
Sandwich        0.037597
Medialuna       0.030039
Hot chocolate   0.028771
Cookies         0.026332
Brownie         0.018481
Name: Item, dtype: float64
```

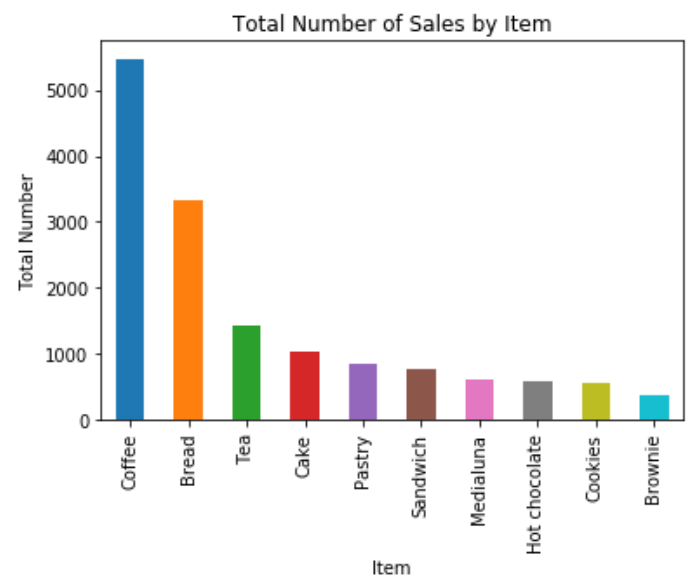
In [10]:
```
# create a bar chart, rank by percentage
bread.Item.value_counts(normalize=True)[:10].plot(kind="bar", title="Percentage of Sales by Item").set(xlabel="Item", y
```

Out[10]: [Text(0,0.5,'Percentage'), Text(0.5,0,'Item')]



In [11]:
```
# create a bar chart, rank by value
bread.Item.value_counts()[:10].plot(kind="bar", title="Total Number of Sales by Item").set(xlabel="Item", ylabel="Total
```

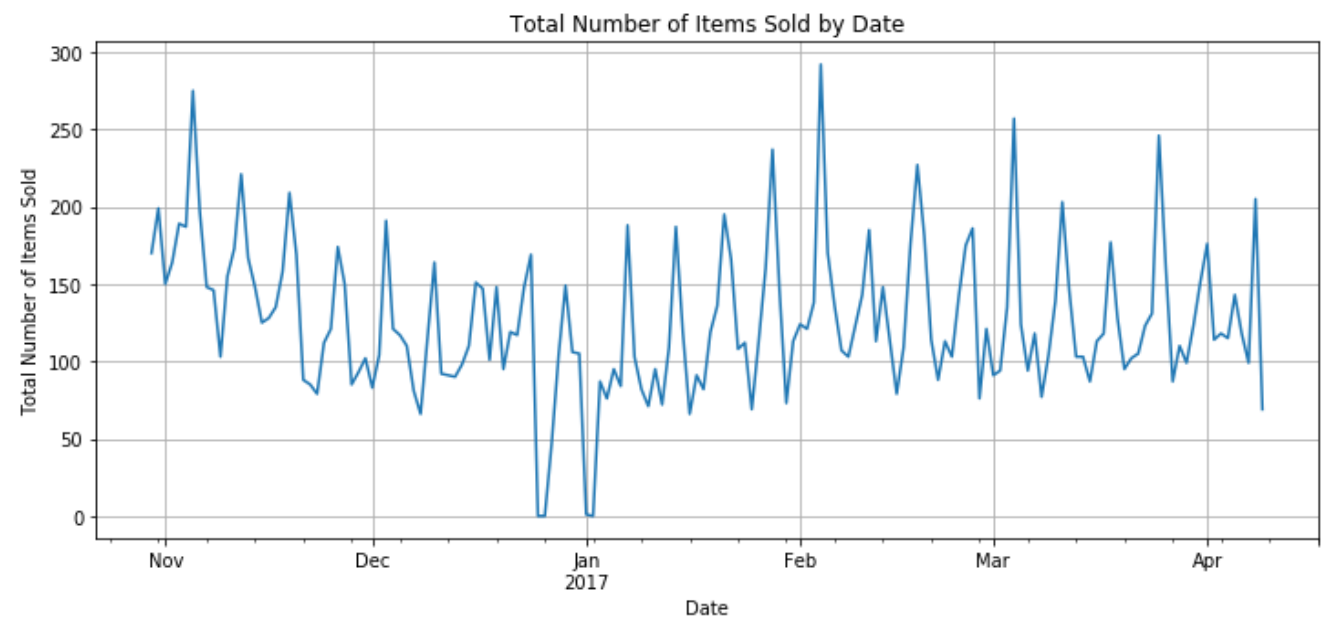
Out[11]: [Text(0,0.5,'Total Number'), Text(0.5,0,'Item')]



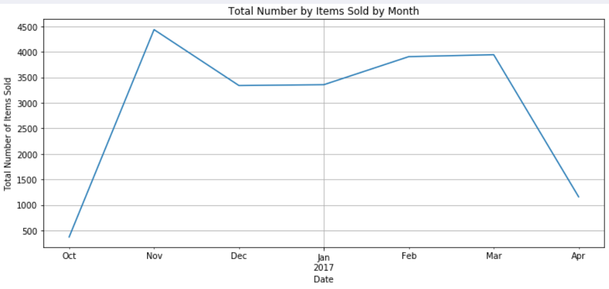
Note: From the bar charts above, it is clear that Coffee (26.7%) is the best-selling item in the bakery, follow by Bread (16.2%) and Tea (7.0%).

In [12]:
```
# plot time series chart of number of items by day
bread["Item"].resample("D").count().plot(figsize=(12,5), grid=True, title="Total Number of Items Sold by Date").set(
```

Out[12]: [Text(0,0.5,'Total Number of Items Sold'), Text(0.5,0,'Date')]



Note: Total Number of Items Sold by Date fluctuates a lot thoughout the 159 days of data



Note: Given that the beginning month (October 2016) and ending month (April 2017) are not full month, the total number of items sold by month for the five full month between November 2016 to March 2017 does not fluctuate too much.

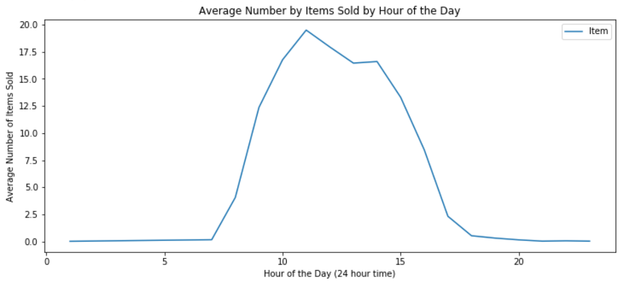
In [15]:
```
# extract hour of the day and weekday of the week
# For Datetimeindex, the day of the week with Monday=0, Sunday=6, thereby +1 to become Monday=1, Sunday=7
bread["Hour"] = bread.index.hour
bread["Weekday"] = bread.index.weekday + 1

bread.head(10)
```

Out[15]:

Datetime	Transaction	Item	Hour	Weekday
2016-10-30 09:58:11	1	Bread	9	7
2016-10-30 10:05:34	2	Scandinavian	10	7
2016-10-30 10:05:34	2	Scandinavian	10	7
2016-10-30 10:07:57	3	Hot chocolate	10	7
2016-10-30 10:07:57	3	Jam	10	7
2016-10-30 10:07:57	3	Cookies	10	7
2016-10-30 10:08:41	4	Muffin	10	7
2016-10-30 10:13:03	5	Coffee	10	7
2016-10-30 10:13:03	5	Pastry	10	7
2016-10-30 10:13:03	5	Bread	10	7

In [17]:
```
# plot the chart
bread_groupby_hour.plot(y="Item", figsize=(12,5), title="Average Number by Items Sold by Hour of the Day").set(xlabel="
```

Out[17]:
```
[Text(0,0.5,'Average Number of Items Sold'),
 Text(0.5,0,'Hour of the Day (24 hour time)')]
```



Note: Sales starts to pick up from 8am, till the busiest hour of the day at 11am, then slowly drops till the late afternoon. It can be observed that most of the sales transactions took place during the lunch hours of the day.

In [18]:
```
# sales groupby weekday
bread_groupby_weekday = bread.groupby("Weekday").agg({"Item": lambda item: item.count()})
bread_groupby_weekday
```

Out[18]:

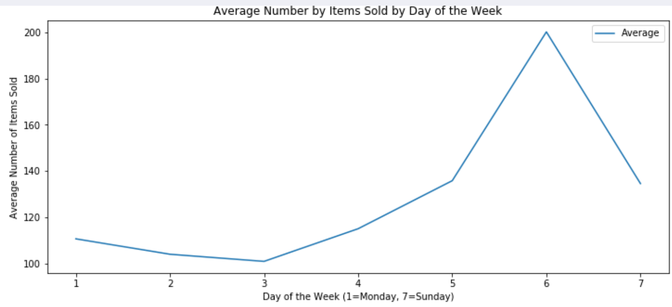
Weekday	Item
1	2324
2	2392
3	2321
4	2646
5	3124
6	4605
7	3095

Out[20]:

Weekday	Item	Average
1	2324	110.666667
2	2392	104.000000
3	2321	100.913043
4	2646	115.043478
5	3124	135.826087
6	4605	200.217391
7	3095	134.565217

In [21]:
```
bread_groupby_weekday.plot(y="Average", figsize=(12,5), title="Average Number by Items Sold by Day of the Week").set(xl
```

Out[21]:
```
[Text(0,0.5,'Average Number of Items Sold'),
 Text(0.5,0,'Day of the Week (1=Monday, 7=Sunday)')]
```



Note: Saturday is the busiest day of the week with the highest sales (~200 items) while Wednesday is the quietest day with the lowest sales (~101 items). This is an interesting insight, the owner of the Bakery should launch some promotion activities to boost up sales in the middle of the week when sales are slowest.

One-Hot Encoding

The **Apriori** function in the MLxtend library expects data in a one-hot encoded pandas DataFrame. This means that all the data for a transaction must be included in one row and the items must be one-hot encoded. Example below:

	Coffee	Cake	Bread	Cookie	Muffin	Tea	Milk	Juice	Sandwich
0	0	1	1	0	0	0	0	1	0
1	1	0	0	0	1	0	0	0	0
2	0	0	0	1	0	0	0	0	1
3	1	0	0	0	0	1	0	0	1
4	1	1	0	0	0	0	0	0	0

Therefore, we'll need to group the bread dataframe by `Transaction` and `Item` and display the count of items. Then we need to consolidate the items into one transaction per row with each item one-hot encoded.

In [22]:

```
df = bread.groupby(["Transaction","Item"]).size().reset_index(name="Count")

df.head()
```

Out[22]:

	Transaction	Item	Count
0	1	Bread	1
1	2	Scandinavian	2
2	3	Cookies	1
3	3	Hot chocolate	1
4	3	Jam	1

In [23]:

```
basket = (df.groupby(['Transaction', 'Item'])['Count']
          .sum().unstack().reset_index().fillna(0)
          .set_index('Transaction'))

basket.head()
```

Out[23]:

Item	Adjustment	Afternoon with the baker	Alfajores	Argentina Night	Art Tray	Bacon	Baguette	Bakewell	Bare Popcorn	Basket	...	The BART	The Nomad	.
Transaction														
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	

5 rows × 94 columns

In [24]:
```
basket[basket.Coffee == 4].iloc[:,14:28]
```

Out[24]:

Item	Brownie	Cake	Caramel bites	Cherry me Dried fruit	Chicken Stew	Chicken sand	Chimichurri Oil	Chocolates	Christmas common	Coffee	Coffee granules	Coke	Cook
Transaction													
6560	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	
6850	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	
6887	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	

Note:

At this stage, the one-hot encoded table shows the count of items purchased as result. If you observe the portion of the table above, in Transaction 6887, the cell value for Coffee is "4.0" because there were 4 coffee purchased in this transaction. However, this is not important for us and we need to convert this value into 1.

In [25]:
```
# the encoding function
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1
```

In [26]:
```
basket_sets = basket.applymap(encode_units)

basket_sets.head()
```

Out[26]:

Item	Adjustment	Afternoon with the baker	Alfajores	Argentina Night	Art Tray	Bacon	Baguette	Bakewell	Bare Popcorn	Basket	...	The BART	The Nomad	.
Transaction														
1	0	0	0	0	0	0	0	0	0	0	...	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	
5	0	0	0	0	0	0	0	0	0	0	...	0	0	

5 rows × 94 columns

In [27]:
```
basket_sets[basket_sets.Coffee == 1].iloc[3142:3145,14:28]
```

Out[27]:

Item	Brownie	Cake	Caramel bites	Cherry me Dried fruit	Chicken Stew	Chicken sand	Chimichurri Oil	Chocolates	Christmas common	Coffee	Coffee granules	Coke	Cook
Transaction													
6884	0	0	0	0	0	0	0	0	0	1	0	0	
6885	1	0	0	0	0	0	0	0	0	1	0	0	
6887	0	1	0	0	0	0	0	0	0	1	0	0	

Note: -

After applying the encoding function, for the same Transaction 6887, the cell value for Coffee has become "1" which is what we need for the **Apriori** function.

Generate Frequent Itemsets

Now, we are ready to generate the frequent item sets. We will set the minimum-support threshold at 1%

In [28]:
```
frequent_itemsets = apriori(basket_sets, min_support=0.01, use_colnames=True)
```

Generate Association Rules

The final step is to generate the rules with their corresponding support, confidence and lift. We will set the minimum threshold for lift at 1 and then sort the result by descending confidence value.

In [29]:
```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.sort_values("confidence", ascending = False, inplace = True)
rules.head(10)
```

Out[29]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
31	(Toast)	(Coffee)	0.033597	0.478394	0.023666	0.704403	1.472431	0.007593	1.764582
29	(Spanish Brunch)	(Coffee)	0.018172	0.478394	0.010882	0.598837	1.251766	0.002189	1.300235
19	(Medialuna)	(Coffee)	0.061807	0.478394	0.035182	0.569231	1.189878	0.005614	1.210871
23	(Pastry)	(Coffee)	0.086107	0.478394	0.047544	0.552147	1.154168	0.006351	1.164682
1	(Alfajores)	(Coffee)	0.036344	0.478394	0.019651	0.540698	1.130235	0.002264	1.135648
16	(Juice)	(Coffee)	0.038563	0.478394	0.020602	0.534247	1.116750	0.002154	1.119919
25	(Sandwich)	(Coffee)	0.071844	0.478394	0.038246	0.532353	1.112792	0.003877	1.115384
7	(Cake)	(Coffee)	0.103856	0.478394	0.054728	0.526958	1.101515	0.005044	1.102664
27	(Scone)	(Coffee)	0.034548	0.478394	0.018067	0.522936	1.093107	0.001539	1.093366
12	(Cookies)	(Coffee)	0.054411	0.478394	0.028209	0.518447	1.083723	0.002179	1.083174

Interpretation and Implications

The output above shows the Top 10 itemsets sorted by confidence value and all itemsets have support value over 1% and lift value over 1. The first itemset shows the association rule "if Toast then Coffee" with support value at 0.023666 means nearly 2.4% of all transactions have this combination of Toast and Coffee bought together. We also have 70% confidence that Coffee sales happen whenever a Toast is purchased. The lift value of 1.47 (greater than 1) shows that the purchase of Coffee is indeed influenced by the purchase of Toast rather than Coffee's purchase being independent of Toast. The lift value of 1.47 means that Toast's purchase lifts the Coffee's purchase by 1.47 times.

Therefore, we can conclude that there is indeed evidence to suggest that the purchase of Toast leads to the purchase of Coffee. The owner of the bakery "The Bread Basket" should consider bundling Toast and Cofee together as a Breakfast Set or Lunch Set, the staff in the store should also be trained to cross-sell Coffee to customers who purchase Toast, knowing that they are more likely to purchase them together, thereby increasing the store's revenue.

Thanks