Open in Colab

Visualizing Data for Classification

In the previous lab, you explored the automotive price dataset to understand the relationships for a regression problem. In this lab you will explore the German bank credit dataset to understand the relationships for a **classification** problem. The difference being, that in classification problems the label is a categorical variable.

In other labs you will use what you learn through visualization to create a solution that predicts the customers with bad credit. For now, the focus of this lab is on visually exploring the data to determine which features may be useful in predicting customer's bad credit.

Visualization for classification problems shares much in common with visualization for regression problems. Colinear features should be identified so they can be eliminated or otherwise dealt with. However, for classification problems you are looking for features that help **separate the label categories**. Separation is achieved when there are distinctive feature values for each label category. Good separation results in low classification error rate.

Load and prepare the data set

Prepare data to a manageable format

Processing bson files

source:

- Kaggle
- Access and process nested objects, arrays or JSON

In [1]:
```
## Unziping (file on linux)
#unzip pantapa_api_development.zip
```

In [2]:
```
##  Convert bson files with, optionally, the outputs documents in a pretty-printed format JSON
#bsondump --pretty --outFile collection.json collection.bson
## OR via https://json-bson-converter.appspot.com/
```

In [3]:
```
## List all the mongodb data .bson files in the dedicated folder

import os

json_arr = os.listdir('data/data-pantapa_bson2json')
print(json_arr)
```

```
['vouchertypes.json', 'scans.json', 'brands.json', 'appinfos.json', 'voucherurls.json', 'vouchers.json', 'vouchertypeurls.json', 'organizations.json', 'materialtypes.json', 'companies.json', 'stations.json', 'prescans.json']
```

In [4]:
```
## Alternatively, proceed as below
## Eg. list all bson files Input data files contained in pantapa_api_development directory

from subprocess import check_output
print(check_output(['ls', 'data/data-pantapa_bson']).decode('utf8'))

# Any results writen to the current directory are saved as output.
```

```
appinfos.bson
appinfos.metadata.json
brands.bson
brands.metadata.json
codenotfounds.metadata.json
companies.bson
companies.metadata.json
materialtypes.bson
materialtypes.metadata.json
modulehashes.metadata.json
organizations.bson
organizations.metadata.json
packages.metadata.json
prescans.bson
prescans.metadata.json
scans.bson
scans.metadata.json
sessiontokens.bson
stations.bson
stations.metadata.json
tokens.bson
tokens.metadata.json
userinformations.metadata.json
vouchers.bson
vouchertypes.bson
vouchertypes.metadata.json
vouchertypeurls.bson
voucherurls.bson
voucherurls.metadata.json
```

In [5]:
```
## Convert files from json to csv, for ease of processing and visualization
import pandas as pd
import json
```

In [6]:
```
## Read and print JSON files into the directory in JSON format
## Let's start with companies

# Open the existing JSON file for loading into a variable
with open('data/data-pantapa_bson2json/companies.json') as json_file:
  companies = json.load(json_file) #This does the same as above, reading the json file and storing it into a variable (dict)

print(companies)
```


```
{'_id': {'machine': -1768797184, 'inc': 299119782, 'time': 1576742726}, 'data': {'name': 'Test', 'active': True, 'alreadyConnected': True, 'show_popup_notification': False}, 'meta': {'timestamp': {'createdAt': 1576742726344, 'updatedAt'
: 1576742726344}}, 'local': {'sv': {'name': 'Test'}}, '__v': 0}
```


In [7]:
```
## Or in pretty json
print(json.dumps(companies, indent=4, sort_keys=True))
```


```
{
    "__v": 0,
    "_id": {
        "inc": 299119782,
        "machine": -1768797184,
        "time": 1576742726
    },
    "data": {
        "active": true,
        "alreadyConnected": true,
        "name": "Test",
        "show_popup_notification": false
    },
    "local": {
        "sv": {
            "name": "Test"
        }
    },
    "meta": {
        "timestamp": {
            "createdAt": 1576742726344,
            "updatedAt": 1576742726344
        }
    }
}
```


In [8]:
```
## Note. We obtain below the same result as when proceeding as above
companies = pd.read_json('data/data-pantapa_bson2json/companies.json', lines=True)
```

In [9]:
```
## Let's convert companies into csv format. We will do the same for the other json files
companies.to_csv (r'data/data-pantapa_json2csv/companies.csv', index = None)
```

In [10]:
```
## Let's check the structure of this newly converted csv file
companies_csv = pd.read_csv('data/data-pantapa_json2csv/companies.csv')

companies.head()
```

Out[10]:

	_id	data	meta	local	__v
0	{'machine': -1768797184, 'inc': 299119782, 'ti...	{'name': 'Test', 'active': True, 'alreadyConne...	{'timestamp': {'createdAt': 1576742726344, 'up...	{'sv': {'name': 'Test'}}	0

Inspecting the data structure for a few of these objects and dictionaries (dict) shows that the csv files do not look like something we want to use for visualization (nested data)... We willl work on json format instead. An easier way could have been to load the bson files on MongoDB, then selecting data subsets of interest for further analysis; we will go straight to that step with the queries down below (in the processing section).

In [11]:
```
## Let's proceed with brands file: read json (already done above) and convert to csv
#brands = pd.read_json('data/data-pantapa_bson2json/brands.json', lines=True)
brands.to_csv (r'data/data-pantapa_json2csv/brands.csv', index = None)
```

```
There was an error when executing cell [11]. Please run Voila with --debug to see the error message.
```

In [12]:
```
print(brands)
#print(json.dumps(brands, indent=4, sort_keys=True))
```

```
There was an error when executing cell [12]. Please run Voila with --debug to see the error message.
```

In [13]:
```
materialtypes = pd.read_json('data/data-pantapa_bson2json/materialtypes.json', lines=True)
materialtypes.to_csv (r'data/data-pantapa_json2csv/materialtypes.csv', index = None)
```

In [14]:
```
organizations = pd.read_json('data/data-pantapa_bson2json/organizations.json', lines=True)
organizations.to_csv (r'data/data-pantapa_json2csv/organizations.csv', index = None)
```

In [15]:
```
prescans = pd.read_json('data/data-pantapa_bson2json/prescans.json', lines=True)
prescans.to_csv (r'data/data-pantapa_json2csv/prescans.csv', index = None)
```

In [16]:
```
scans = pd.read_json('data/data-pantapa_bson2json/scans.json', lines=True)
scans.to_csv (r'data/data-pantapa_json2csv/scans.csv', index = None)
```

```
There was an error when executing cell [16]. Please run Voila with --debug to see the error message.
```

In [17]:
```
stations = pd.read_json('data/data-pantapa_bson2json/stations.json', lines=True)
stations.to_csv (r'data/data-pantapa_json2csv/stations.csv', index = None)
```

In [18]:
```
vouchertypes = pd.read_json('data/data-pantapa_bson2json/vouchertypes.json', lines=True)
vouchertypes.to_csv (r'data/data-pantapa_json2csv/vouchertypes.csv', index = None)
```

In [19]:
```
vouchertypeurls = pd.read_json('data/data-pantapa_bson2json/vouchertypeurls.json', lines=True)
vouchertypeurls.to_csv (r'data/data-pantapa_json2csv/vouchertypeurls.csv', index = None)
```

In [20]:
```
voucherurls = pd.read_json('data/data-pantapa_bson2json/voucherurls.json', lines=True)
voucherurls.to_csv (r'data/data-pantapa_json2csv/voucherurls.csv', index = None)
```

Let's check the list of converted csv files:

In [21]:
```
from subprocess import check_output
print(check_output(['ls', 'data/data-pantapa_json2csv']).decode('utf8'))
```

```
brands.csv
companies.csv
materialtypes.csv
organizations.csv
prescans.csv
scans.csv
stations.csv
vouchers.csv
vouchertypes.csv
vouchertypeurls.csv
voucherurls.csv
```

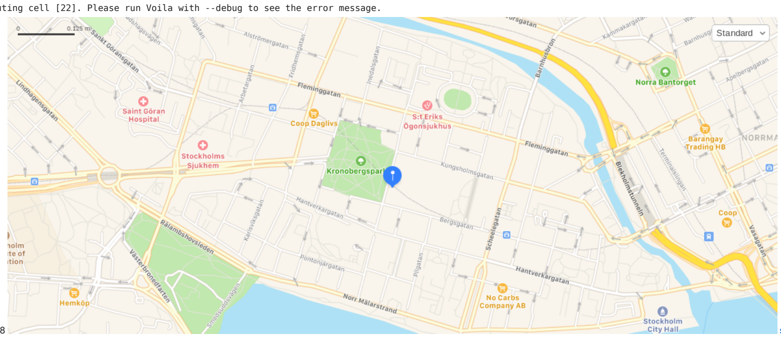
Extract objects from nested JSON

and explore datasets

In [22]:
```
## Inspect content of the scans dictionary
print(scans['data']['enums']['location']['coordinates'])
```

```
There was an error when executing cell [22]. Please run Voila with --debug to see the error message.
```

Lat = 59.3313148, Long = 18.0373788



source

In [23]:
```
print(scans['data']['enums']['name'])
```

```
There was an error when executing cell [23]. Please run Voila with --debug to see the error message.
```

Let's proceed the same way with the other dictionaries obtained above from reading json files.

In [24]:
```
print(prescans['data']['enums']['location']['coordinates'])
```

```
There was an error when executing cell [24]. Please run Voila with --debug to see the error message.
```

In [25]:
```
print(prescans['data']['enums']['status'])
```

```
There was an error when executing cell [25]. Please run Voila with --debug to see the error message.
```

In [26]:
```
print(prescans['data']['enums']['name'])
```

```
There was an error when executing cell [26]. Please run Voila with --debug to see the error message.
```

In [27]:
```
print(vouchers['data']['redeem_date'])
```

```
There was an error when executing cell [27]. Please run Voila with --debug to see the error message.
```

In [28]:
```
print(vouchers['data']['coupon']['validTo'])
```

```
There was an error when executing cell [28]. Please run Voila with --debug to see the error message.
```

In [29]:
```
print(vouchers['data']['coupon']['name'])
```

```
There was an error when executing cell [29]. Please run Voila with --debug to see the error message.
```

In [30]:
```
print(vouchers['data']['coupon']['htmlLink'])
```

```
There was an error when executing cell [30]. Please run Voila with --debug to see the error message.
```

In [31]:
```
print(vouchers['data']['coupon']['couponCode'])
```

```
There was an error when executing cell [31]. Please run Voila with --debug to see the error message.
```

We observe that there is only one brand in this file. Not enough to draw any pattern or trend, yet interersting to explore in depth some variables of interest for information purpose. To get to know the data better.

In [32]:
```
print(brands['data']['name'])
```

```
There was an error when executing cell [32]. Please run Voila with --debug to see the error message.
```

In [33]:
```
print(brands['data']['image']['source'])
```

```
There was an error when executing cell [33]. Please run Voila with --debug to see the error message.
```

apoteket_logo

In [34]:
```
print(brands['data']['country_code'])
```

There was an error when executing cell [34]. Please run Voila with --debug to see the error message.

In [35]:
```
print(brands['local']['sv']['how_it_works']['image_link']['source'])
```

There was an error when executing cell [35]. Please run Voila with --debug to see the error message.

Bags_Green

In [36]:
```
print(brands['local']['sv']['how_it_works']['text_line1_url'])
```

There was an error when executing cell [36]. Please run Voila with --debug to see the error message.

In [37]:
```
print(brands['local']['sv']['how_it_works']['package_name'])
```

There was an error when executing cell [37]. Please run Voila with --debug to see the error message.

In [38]:
```
print(brands['local']['en']['how_it_works']['description'])
```

There was an error when executing cell [38]. Please run Voila with --debug to see the error message.

In [39]:
```
## From the Read and print JSON file in JSON format previous steps,

# Also equivalent to what obtained by the queries below (opening the existing JSON file for loading into a variable)
#with open('data/data-pantapa_bson2json/stations.json') as json_file:
#  stations = json.load(json_file)

## Let's print pretty JSON data
print(json.dumps(stations, indent=4, sort_keys=True))
```

There was an error when executing cell [39]. Please run Voila with --debug to see the error message.

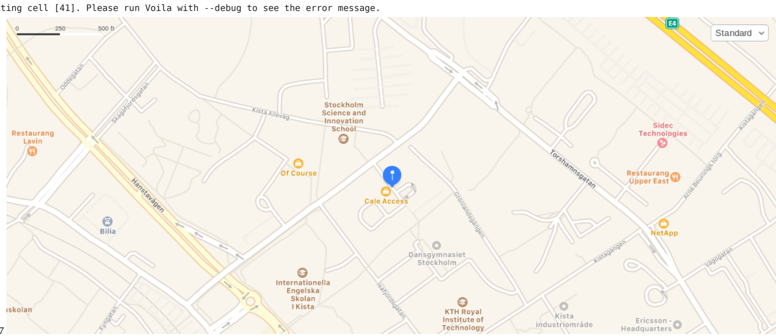
As usual, let's look closer at the data

In [40]:
```
print(stations['data']['address'])
```

There was an error when executing cell [40]. Please run Voila with --debug to see the error message.

In [41]:
```
print(stations['data']['location']['point_type'])
```

There was an error when executing cell [41]. Please run Voila with --debug to see the error message.



Lat = 59.4067509, Long = 17.9472797

In [42]:
```
print(stations['data']['point_type'])
```

There was an error when executing cell [42]. Please run Voila with --debug to see the error message.

In [43]:
```
print(stations['data']['scan_distance'])
```

There was an error when executing cell [43]. Please run Voila with --debug to see the error message.

In [44]:
```
print(stations['data']['store'])
```

There was an error when executing cell [44]. Please run Voila with --debug to see the error message.

In [45]:
```
print(organizations['_id']['machine'])
```

There was an error when executing cell [45]. Please run Voila with --debug to see the error message.

In [46]:
```
print(organizations['data']['name'])
```

There was an error when executing cell [46]. Please run Voila with --debug to see the error message.

As datasets do not have significant numbers of products to visualize, extract patterns and/or predict future behaviours (such as articles often bought, i.e scanned, together), we will do the predictive analytics work on a dummy dataset of choice. For this purpose, we will put ourselves in the situation where ALL products are scannable. The method of choice we will implement is call **Market Basket Analysis**

Market Basket Analysis (MBA)

In a first part, we will briefly explain the MBA basics and illustrate it with a case study of items scanned in a supermarket. In the second part we will implement this technique in python language programming using public dataset from model some source coded on github. Alternatively, this other dataset can be used for a more visual analysis.
References at the end of this project.

source

Understanding MBA

In this hypothetical case study, we are going to use the Apriori algorithm for frequent pattern mining to perform a Market Basket Analysis. Following sources (Xavier Vivancos García), "MBA is a technique used by large retailers to *uncover associations between items*. It works by looking for combinations of items that occur together frequently in transactions, providing information to understand the purchase behavior. The outcome of this type of technique is, in simple terms, a set of rules that can be understood as "if this, then that"."

Additional sources (limchiahooi), define "Market basket analysis (MBA), also known as association-rule mining, as a method of discovering *customer purchasing patterns* by extracting *associations or co-occurrences* from stores' transactional databases. It is a modelling technique based upon the theory that if you buy a certain group of items, you are more (or less) likely to buy another group of items. For example, 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. (...)"
Same principle can in theory be applied to *scanned items* -- as the scanning process is an integrated part of the purchasing process.

Applications

There are many real-life applications of MBA:

- 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.
- 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.
- 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.

  Etc.

Case Study

We are analyzing the hypothetic scanning case of two items – Bread and Butter. We want to know if there is any evidence that suggests that scanning Bread leads to scanning Butter. Note. We will often replace scanning by transaction, interchangeably.

Problem Statment: Is the pscanning of Bread leads to the scanning of Butter?

Hypothesis: There is significant evidence to show that scanning Bread leads to scanning Butter. (As much as buying Bread leads to buying Butter)

Bread => Butter

Antecedent => Consequent

Let's consider 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.

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. (*Convictions* is an additional metric used in some cases)
More about this here

In short,

- Support measures the percentage of transactions containing a particular combination of items relative to the total number of transactions.
  In our example: *Support (antecedent (Bread) and consequent (Butter)) = Number of transactions having both items / Total transactions.*
  Result: The support value of 5% means 5% of all transactions have this combination of Bread and Butter scanned 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.*

  $$\begin{aligned} &\text{P (Bread INTERSECTION Butter)} \\ &= \text{P (Bread} \cap \text{Butter)} \\ &= \frac{\text{Number of transactions with Bread AND Butter}}{\text{Total transactions}} \\ &= \frac{50}{1000} \\ &= 5\% \end{aligned}$$

- Confidence measures the probability of finding a particular combination of items whenever antecedent is bought.
  Confidence (antecedent i.e. Bread and consequent i.e. Butter) = P (Consequent (Butter) is bought GIVEN antecedent (Bread) is bought).

  $$\begin{aligned} &\text{P (Butter GIVEN Bread)} \\ &= \frac{\text{P (Bread} \cap \text{Butter)}}{\text{P (Bread)}} \\ &= \frac{\text{Number of transactions with Bread AND Butter}}{\text{Number of transactions with Bread}} \\ &= \frac{50}{150} \\ &= 33.3\% \end{aligned}$$

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

- Lift is a metric to determine how much the transaction between antecedent and consequent influence each other.
  We want to know which is higher, P(Butter) or P(Butter / Bread)? (Conditional probabilities) If the scanning of Butter is influenced by the one of Bread, then the *ratio of P(Butter / Bread) over P(Butter) > 1.*
  Result: The lift value of 2.56 is greater than 1, thus that the transaction for Butter is indeed influenced by the one for Bread which *satisfy the third criteria*. This also means that Bread's transaction lifts the Butter's purchase by 2.56 times.

Takeaways

Based on the findings above, we

```
a) Have the support of 5% transactions for Bread and Butter in the same basket
b) Have 33.3% confidence that Butter scan happen whenever Bread is scanned.
c) Know the lift in Butter's transaction is 2.56 times more whenever Bread is involved than when Butter is alone.
```

Therefore, we can justify our initial hypothesis by concluding that there is indeed evidence to suggest that the *transaction for Bread leads to the one for Butter*. This is a valuable insight to guide decision-making.
Actions forward could be, among other things, for retail stores to start placing bread and butter close to each other, knowing that customers are highly likely to "impulsively" scanned (and ultimately purchase) them together.

Implementation in Python

On a large dataset, leveraging on Python libraries for a ready-made algorithm is more efficient than the use of traditional Ms Excel to calculate support, confidence and lifts. Furthermore, as the popular scikit-learn library does not allow us to apply *Apriori algorithm* for extracting frequent item sets for further analysis, because not supported this algorithm, we use another library instead: MLxtend (machine learning extensions) by Sebastian Raschka. Chris Moffitt also provides a tutorial on using MLxtend.

Note. If you are using Jupyter Notebook, the MLxtend library does not come pre-installed with Anaconda (which I am using right now). You can easily install this package with conda by running one of the following in your Anaconda Prompt:

```
conda install -c conda-forge mlxtend
conda install -c conda-forge/label/gcc7 mlxtend
```

Or with pip:

```
!pip install mlxtend
```

("!" if cell ran from the notebook)

Dataset

The dataset we are using in the case study in this is inspired from a publicly available one initially from Kaggle, now hosted on github which contains the Transactions data from a bakery from 30/10/2016 to 09/04/2017. The original data belongs to a real bakery called "The Bread Basket" that serves coffee, bread, muffin, cookies etc. located in the historic center of Edinburgh.

Import libraries

In [47]:
```
#!pip install mlxtend
```

In [48]:
```
# import the libraries required
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

Load data

In [49]:
```
# load the data into a pandas dataframe (df) and take a look at the first 10 rows
df = pd.read_csv("https://raw.githubusercontent.com/limchiahooi/market-basket-analysis/master/BreadBasket_DMS.csv")

# Let's rename Transaction column with Scanned just to be more representative of our thought experiment
df_new = df.rename(columns={'Transaction': 'Scanned'})

df_new.head(5)
```

Out[49]:

	Date	Time	Scanned	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

In [50]:
```
# Date and Time are encoded in 'object' instead of Datetime
df_new['Datetime'] = pd.to_datetime(df_new['Date']+' '+df_new['Time'])

df_new = df_new[["Datetime", "Scanned", "Item"]].set_index("Datetime")

df_new.head(10)
```

Out[50]:

Datetime	Scanned	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

We have combined the Date and Time columns into a single Datetime column, convert it into datetime64 type, then set it as DatetimeIndex. This will make it easier to plot the time series charts.

In [51]:
```
# let's check the shape of the dataset.
## 18768 rows and 2 columns
df_new.shape
```

Out[51]: (21293, 2)

In [52]:
```
df_new.describe()
```

Out[52]:

	Scanned
count	21293.000000
mean	4951.990889
std	2787.758400
min	1.000000
25%	2548.000000
50%	5067.000000
75%	7329.000000
max	9684.000000

In [53]:
```
missing_value = ["NaN", "NONE", "None", "Nil", "nan", "none", "nil", 0]

print("There are {} missing values in the dataframe.".format(len(df_new[df_new.Item.isin(missing_value)])))
df_new[df_new.Item.isin(missing_value)].head(5)
```

There are 786 missing values in the dataframe.

Out[53]:

Datetime	Scanned	Item
2016-10-30 10:27:21	11	NONE
2016-10-30 10:34:36	15	NONE
2016-10-30 10:34:36	15	NONE
2016-10-30 11:05:30	29	NONE
2016-10-30 11:37:10	37	NONE

Since the items (NONE) are not recorded, we will have to remove these rows.

In [54]:
```
df_new = df_new.drop(df_new[df_new.Item == "NONE"].index)
print("Number of rows: {}".format(len(df_new)))

df_new.head(5)
```

```
Number of rows: 18768
```

Out[54]:

	Scanned	Item
Datetime		
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

Visualization

In [55]:

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

Out[55]:

```
Coffee           0.264226
Bread            0.165974
Tea              0.066922
Cake             0.052003
Pastry           0.043212
Sandwich         0.036125
Medialuna        0.030744
Hot chocolate    0.028186
Cookies          0.027227
Farm House       0.019288
Name: Item, dtype: float64
```

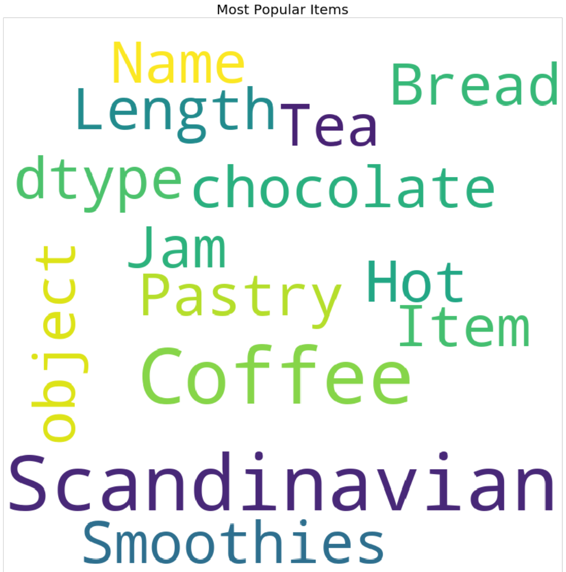
In [56]:

```
## In order to plot a word cloud, without capturing the DateTime index, we drop it and save it into a new dataframe

df1 = df_new.reset_index()
#df1[["Scanned", "Item"]]
#df1
```

In [57]:

```
## Ploting a word cloud of the most popular items

import matplotlib.pyplot as plt
import seaborn as sns

from wordcloud import WordCloud

plt.rcParams['figure.figsize'] = (15, 15)
wordcloud = WordCloud(background_color = 'white', width = 800,  height = 800, max_words = 121).generate(str(df1["Item"]))
plt.imshow(wordcloud)
plt.axis('off')
plt.title('Most Popular Items',fontsize = 20)
plt.show()
```

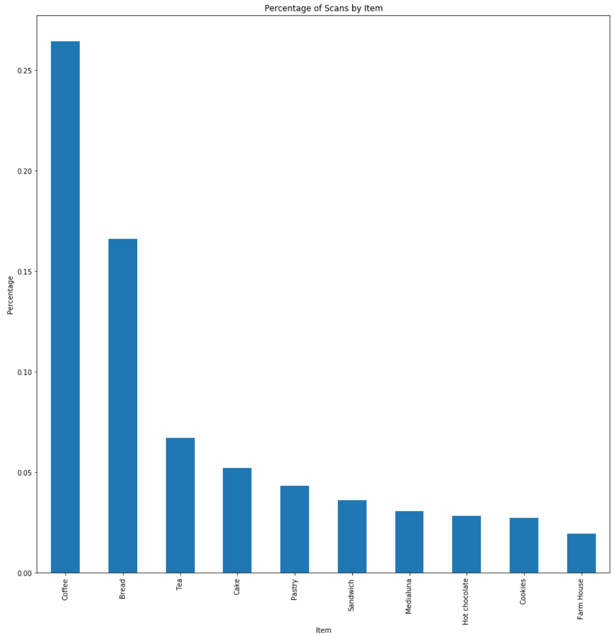
Most Popular Items



In [58]:

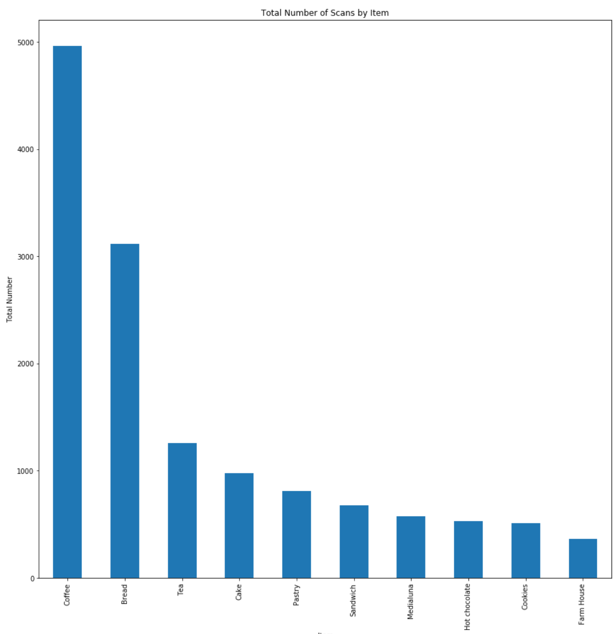
```
# create a bar chart, rank by percentage
df_new.Item.value_counts(normalize=True)[:10].plot(kind="bar", title="Percentage of Scans by Item").set(xlabel="Item", ylabel="Percentage")
```

```
Out[58]: [Text(0,0.5,'Percentage'), Text(0.5,0,'Item')]
```




```
In [59]: # create a bar chart, rank by value
df_new.Item.value_counts()[:10].plot(kind="bar", title="Total Number of Scans by Item").set(xlabel="Item", ylabel="Total Number")
```

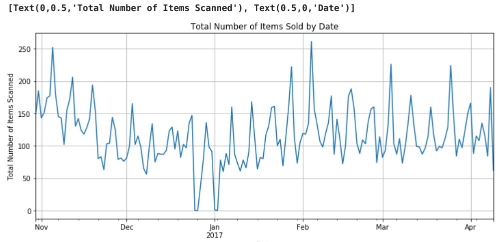
```
Out[59]: [Text(0,0.5,'Total Number'), Text(0.5,0,'Item')]
```



From the bar charts above, Coffee (26.7%) is the most popular item in the bakery, then Bread (16.2%) followed with Tea (7.0%).

```
In [60]: # plot time series chart of number of items by day
df_new["Item"].resample("D").count().plot(figsize=(12,5), grid=True, title="Total Number of Items Sold by Date").set(xlabel="Date", ylabel="Total Number of Items Scanned")
```

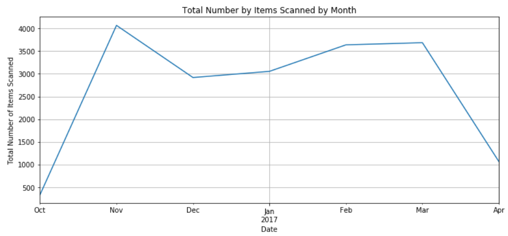
```
Out[60]: [Text(0,0.5,'Total Number of Items Scanned'), Text(0.5,0,'Date')]
```



Total Number of Items Scanned by Date fluctuates thoughout the 159 days of the data time frame.

```
In [61]: # plot time series chart of number of items by month
df_new["Item"].resample("M").count().plot(figsize=(12,5), grid=True, title="Total Number by Items Scanned by Month").set(xlabel="Date", ylabel="Total Number of Items Scanned")
```

Out[61]:
```
[Text(0,0.5,'Total Number of Items Scanned'), Text(0.5,0,'Date')]
```



The total number of items scanned by month for the five full month between November 2016 to March 2017 does not fluctuate too much.

In [62]:
```python
# 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
df_new["Hour"] = df_new.index.hour
df_new["Weekday"] = df_new.index.weekday + 1

df_new.head(5)
```

Out[62]:

Datetime	Scanned	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

In [63]:
```python
df_new_groupby_hour = df_new.groupby("Hour").agg({"Item": lambda item: item.count()/total_days})
df_new_groupby_hour
```

```
There was an error when executing cell [63]. Please run Voila with --debug to see the error message.
```

In [64]:
```python
# plot the chart
df_new_groupby_hour.plot(y="Item", figsize=(12,5), title="Average Number by Items Scanned by Hour of the Day").set(xlabel="Hour of the Day (24 hour time)", ylabel="Average Number of Items Scanned")
```

```
There was an error when executing cell [64]. Please run Voila with --debug to see the error message.
```

It appears that most of the sales transactions took place during the lunch hours of the day.

In [65]:
```python
# sales groupby weekday
df_new_groupby_weekday = df_new.groupby("Weekday").agg({"Item": lambda item: item.count()})
df_new_groupby_weekday
```

Out[65]:

Weekday	Item
1	2196
2	2261
3	2099
4	2485
5	2931
6	4098
7	2698

In [66]:
```python
# but we need to find out how many each weekday in that period of transaction
# in order to calculate the average items per weekday

import datetime
daterange = pd.date_range(datetime.date(2016, 10, 30), datetime.date(2017, 4, 9))

monday = 0
tuesday = 0
wednesday = 0
thursday = 0
friday = 0
saturday = 0
sunday = 0

for day in np.unique(df_new.index.date):
    if day.isoweekday() == 1:
        monday += 1
    elif day.isoweekday() == 2:
        tuesday += 1
    elif day.isoweekday() == 3:
        wednesday += 1
    elif day.isoweekday() == 4:
        thursday += 1
    elif day.isoweekday() == 5:
        friday += 1
    elif day.isoweekday() == 6:
        saturday += 1
    elif day.isoweekday() == 7:
        sunday += 1

all_weekdays = monday + tuesday + wednesday + thursday + friday + saturday + sunday

print("monday = {0}, tuesday = {1}, wednesday = {2}, thursday = {3}, friday = {4}, saturday = {5}, sunday = {6}, total = {7}".format(monday, tuesday, wednesday, thursday, friday, saturday, sunday, all_weekdays))
```

```
monday = 21, tuesday = 23, wednesday = 23, thursday = 23, friday = 23, saturday = 23, sunday = 23, total = 159
```

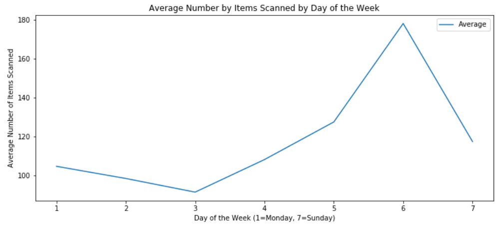
In [67]:
```python
# apply the conditions to calculate the average items for each weekday
conditions = [
    (df_new_groupby_weekday.index == 1),
    (df_new_groupby_weekday.index == 2),
    (df_new_groupby_weekday.index == 3),
    (df_new_groupby_weekday.index == 4),
    (df_new_groupby_weekday.index == 5),
    (df_new_groupby_weekday.index == 6),
    (df_new_groupby_weekday.index == 7)]

choices = [df_new_groupby_weekday.Item/21, df_new_groupby_weekday.Item/23, df_new_groupby_weekday.Item/23, df_new_groupby_weekday.Item/23, df_new_groupby_weekday.Item/23, df_new_groupby_weekday.Item/23, df_new_groupby_weekday.Item/23]

df_new_groupby_weekday["Average"] = np.select(conditions, choices, default=0)
df_new_groupby_weekday
```

Out[67]:

Weekday	Item	Average
1	2196	104.571429
2	2261	98.304348
3	2099	91.260870
4	2485	108.043478
5	2931	127.434783
6	4098	178.173913
7	2698	117.304348

In [68]:
```python
df_new_groupby_weekday.plot(y="Average", figsize=(12,5), title="Average Number by Items Scanned by Day of the Week").set(xlabel="Day of the Week (1=Monday, 7=Sunday)", ylabel="Average Number of Items Scanned")
```

Out[68]:
```
[Text(0,0.5,'Average Number of Items Scanned'),
 Text(0.5,0,'Day of the Week (1=Monday, 7=Sunday)')]
```



We observe Saturday is the busiest day of the week with the highest transactions while Wednesday is the quietest day with the lowest ones. This is an interesting insight, the store owner could for eg. launch some promotional offers to boost up activity in the middle of the week when level is at its 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` (row) and `Item` (column) 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 [69]:
```
df_new = df_new.groupby(["Scanned","Item"]).size().reset_index(name="Count")

df_new.head()
```

Out[69]:

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

In [70]:
```
MBA = (df_new.groupby(['Scanned', 'Item'])['Count']
          .sum().unstack().reset_index().fillna(0)
          .set_index('Scanned'))

MBA.head()
```

Out[70]:

Item	Adjustment	Afternoon with the baker	Alfajores	Argentina Night	Art Tray	Baguette	Bakewell	Bare Popcorn	Basket	Bowl Nic Pitt	...	The BART	The Nomad	Tiffin	Toast	Truffles	Tshirt	Valentine's card	Vegan Feast	Vegan mincepie	Victorian Sponge
Scanned																					
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	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	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	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	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 93 columns

In [71]:
```
MBA[MBA.Coffee == 4].iloc[:,14:28]
#MBA[MBA.Bread == 4].iloc[:,1:14]
```

Out[71]:

Item	Cake	Caramel bites	Cherry me Dried fruit	Chicken Stew	Chicken sand	Chimichurri Oil	Chocolates	Christmas common	Coffee	Coffee granules	Coke	Cookies	Crepes	Crisps
Scanned														
6560	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	1.0	0.0	0.0
6850	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
6887	1.0	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

Note. In Transaction 6887 for eg. the cell value for Coffee is "4.0" because there were 4 coffee purchased in this transaction. However, this is not important here as we are mainly interested by transactions below, equal or above 0 and 1. We will thus encode values into 1. After applying the encoding function, for the same Transaction 6887, the cell value for Coffee should become "1" which is what we need for the Apriori function.

In [72]:
```
# the encoding function
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1
```

In [73]:
```
MBA_sets = MBA.applymap(encode_units)

MBA_sets.tail()
```

Out[73]:

Item	Adjustment	Afternoon with the baker	Alfajores	Argentina Night	Art Tray	Baguette	Bakewell	Bare Popcorn	Basket	Bowl Nic Pitt	...	The BART	The Nomad	Tiffin	Toast	Truffles	Tshirt	Valentine's card	Vegan Feast	Vegan mincepie	Victorian Sponge
Scanned																					
9680	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9681	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	0	0
9682	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9683	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9684	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 93 columns

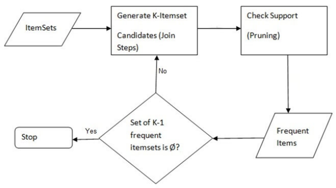
In [74]:
```
MBA_sets[MBA_sets.Coffee == 1].iloc[3142:3145,14:28]
```

Out[74]:

Item	Cake	Caramel bites	Cherry me Dried fruit	Chicken Stew	Chicken sand	Chimichurri Oil	Chocolates	Christmas common	Coffee	Coffee granules	Coke	Cookies	Crepes	Crisps
Scanned														
7532	0	0	0	0	0	0	0	0	1	0	0	0	0	0
7535	0	0	0	0	0	0	0	0	1	0	0	0	0	0
7536	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Applying Apriori algorithm and Association Rules

Apriori algorithm logical diagram



source

How does Apriori Algorithm Work ?

A key concept in Apriori algorithm is the anti-monotonicity of the support measure. It assumes that

- **All subsets of a frequent itemset must be frequent**
- **Similarly, for any infrequent itemset, all its supersets must be infrequent too**

Step 1: Create a frequency table of all the items that occur in all the transactions.

Step 2: We know that only those elements are significant for which the support is greater than or equal to the threshold support.

Step 3: The next step is to make all the possible pairs of the significant items keeping in mind that the order doesn't matter, i.e., AB is same as BA.

Step 4: We will now count the occurrences of each pair in all the transactions.

Step 5: Again only those itemsets are significant which cross the support threshold

Step 6: Now let's say we would like to look for a set of three items that are purchased together. We will use the itemsets found in step 5 and create a set of 3 items.

Frequent Itemsets

Now that we have hot-encoded all the values above 1 into 1, we are ready to generate the frequent item sets. We will set the minimum-support threshold at 1%

In [75]:
```
frequent_itemsets = apriori(MBA_sets, min_support=0.01, use_colnames=True)
```

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 [76]:
```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.sort_values("confidence", ascending = False, inplace = True)
rules.head(10)
```

Out[76]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
29	(Toast)	(Coffee)	0.032126	0.470153	0.022442	0.698582	1.485861	0.007338	1.757846
19	(Medialuna)	(Coffee)	0.062315	0.470153	0.034974	0.561243	1.193747	0.005676	1.207610
23	(Pastry)	(Coffee)	0.087833	0.470153	0.048075	0.547341	1.164177	0.006780	1.170522
1	(Alfajores)	(Coffee)	0.037138	0.470153	0.019936	0.536810	1.141778	0.002476	1.143909
25	(Sandwich)	(Coffee)	0.067555	0.470153	0.035657	0.527825	1.122666	0.003896	1.122141
7	(Cake)	(Coffee)	0.106516	0.470153	0.055366	0.519786	1.105569	0.005287	1.103357
17	(Juice)	(Coffee)	0.034860	0.470153	0.017772	0.509804	1.084337	0.001382	1.080889
13	(Cookies)	(Coffee)	0.055594	0.470153	0.028139	0.506148	1.076560	0.002001	1.072886
27	(Scone)	(Coffee)	0.035202	0.470153	0.017772	0.504854	1.073809	0.001222	1.070084
15	(Hot chocolate)	(Coffee)	0.056277	0.470153	0.027797	0.493927	1.050568	0.001338	1.046978

Interpretation and Implications

From the table above, we observe that the Top 10 itemsets sorted by confidence value and all itemsets have support value over 1% and lift value over 1.
As we have focused on Coffee due to its value of 4 converted into 1 (as an illustration of one hot-encoding technique), we will continue with the exploration this item: The first itemset shows the association rule "if Toast then Coffee" with *support value at 0.022442 means nearly 2.4% of all transactions have this combination of Toast and Coffee bought together*. We also have roughly *70% confidence* that Coffee sales happen whenever a Toast is purchased. The *lift value of 1.48* (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 1.48 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.
Same analysis can be performed for Bread and Butter, as initially aimed.

The owner of the bakery "The Bread Basket" should, as an actionable decision forward for example, 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.

References

- **Amir, A. (2019, February 3). Association Rule(Apriori and Eclat Algorithms) with Practical Implementation. *Medium*. Retrieved from** https://medium.com/machine-learning-researcher/association-rule-apriori-and-eclat-algorithm-4e963fa972a4
- **Kaushik, D. (2019, January 15). Product Recommendation Case Study Using Apriori Algorithm for a Grocery Store. *Medium*. Retrieved from** https://medium.com/datadriveninvestor/product-recommendation-using-association-rule-mining-for-a-grocery-store-7e7feb6cd0f9
- **Madalina, C. (2019, Juin 8). An introduction to frequent pattern mining research. Summary of Apriori, Eclat and FP tree algorithms. *IMedium*. Retrieved from** https://medium.com/@ciortanmadalina/an-introduction-to-frequent-pattern-mining-research-564f239548e
- **Andrewngai (2020, March 17). Understand and Build FP-Growth Algorithm in Python. Frequency Pattern Mining using FP-tree and conditional FP-tree in Python. *Towards Data Science*. Retrieved from** https://towardsdatascience.com/understand-and-build-fp-growth-algorithm-in-python-d8b989bab342
- **Xavier Vivancos, G. (2020, May). Market Basket Analysis. *Kaggle*. Retrieved from** https://www.kaggle.com/xvivancos/market-basket-analysis

In [ ]: