University of Salford, MSc Data Science

Module: Machine Learning & Data Mining

Date: Trimester 1, 2024-2025

Session: Workshop Week 3

Topic: Association Rules Mining

Tools: Jupyter Notebook

Objectives:

After completing this workshop, you will be able to:

- Understand how to interpret confidence, support, and lift
- Use the apyori Python package to run association rules mining using the apriori algorithm
- Analyze and interpret the association rules you generate

MLDM Week 3 Workshop: Association Rule Mining

Introduction

Association rule mining is the data mining process of finding the rules that may govern associations between sets of items. The term market basket analysis refers to a specific implementation of association rules mining that many companies use for a variety of purposes, so in a given transaction with multiple items, it tries to find the rules that govern how or why such items are often bought together. Association rules are rules which surpass a user-specified minimum Support and minimum Confidence threshold. For example, diapers and beer are often purchased together because, as it turns out, dads are often tasked with shopping while the moms take care of the baby.

The main applications of association rule mining are:

- Basket data analysis - is to analyse the association of purchased items in a single basket or a single purchase.
- Cross marketing - is to work with other businesses that complement your own, not competitors.
- Catalogue design - the selection of items in a business' catalogue are often designed to complement each other so that buying one item will lead to buying of another. So, these items are often complements or very related.

Besides market basket analysis, association rules are commonly used for recommender systems and click stream analysis. Many online service providers such as Amazon and Netflix use recommender systems. Recommender systems can use association rules to discover related products or identify customers who have similar interests. For example, association rules may suggest that those customers who have bought product A have also bought product B, or those customers who have bought products A, B, and C are

more similar to this customer. These findings provide opportunities for retailers to cross-sell their products. Click stream analysis refers to the analytics on data related to web browsing and user clicks, which is stored on the client or the server side. Web usage log files generated on web servers contain huge amounts of information, and association rules can potentially give useful knowledge to web usage data analysts.

Support:

The support, supp(X), of an item set X is defined as the proportion of transactions in the dataset which contain the item set. For example, the item set {milk, bread}in Figure 1. has a support of 2/5 = 0.4 since it occurs in 40% of all transactions (2 out of 5 transactions).

transaction ID	items
1	milk, bread
2	bread, butter
3	beer
4	milk, bread, butter
5	bread, butter

Figure 1: An example of supermarket database with five transactions

$$support(B) = \frac{Transactions\ containing\ B}{Total\ Transactions}$$

Confidence:

Confidence of a rule is defined as conf (X ⇒ Y) = supp(X ∪ Y)/supp(X). For example, the rule {milk, bread} ⇒ {butter} has a confidence of 0.2/0.4 = 0.5 in the database in Figure 1, which means that for 50% of the transactions containing milk and bread the rule is correct. Confidence can be interpreted as an estimate of the probability P(Y |X), the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS.

$$confidence(A \rightarrow B) = \frac{Transactions\ containing\ A\ and\ B}{Transactions\ containing\ A}$$

Lift:

Lift is a measure of the strength of the association between two items, taking into account the frequency of both items in the dataset. We define lift as the ratio of the observed support measure and expected support if antecedent and consequent itemsets are independent of each other.

The lift of a rule is defined as:

$$lift(A \rightarrow B) = \frac{support(A \; and \; B)}{support(A) \times support(B)}$$

For example, imagine that milk and bread are both bought by customers 50% of the time. In this case, if there was no association between milk and bread, you would still expect them to be found together in a customer's basket around 25% (50%*50%) of the time. This would therefore give a support of 25% and a confidence of 50%, even if there was no association between the two. The lift is therefore an important measure to understand how strong the association between items in the rule is. A lift of 1 suggests that the items are co-occurring no more than expected if there were no association between them. A lift of 2 suggests that the items are co-occurring twice as often as would be expected if there was no association between them. Greater lift values (>> 1) indicate stronger associations.

Important Note: Saving Your Notebook to F Drive

By default, when you launch Jupyter Notebook, the starting directory is in the C drive. If you are using a university desktop or laptop to complete this workshop, you should make sure that at the end of session you save your notebook to F drive so you can access it from other university devices.

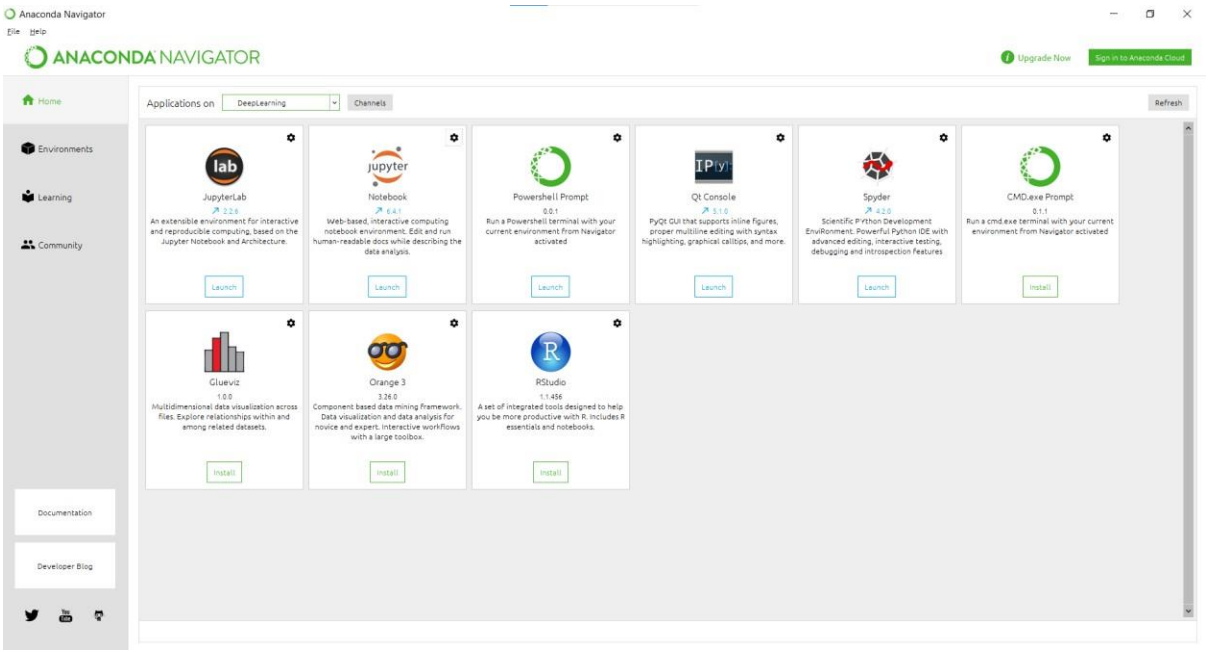
Alternatively, you can launch Jupyter Notebook with the F drive as the starting directory by opening the Anaconda Prompt and using the below command:

```
jupyter notebook --notebook-dir=F:\
```

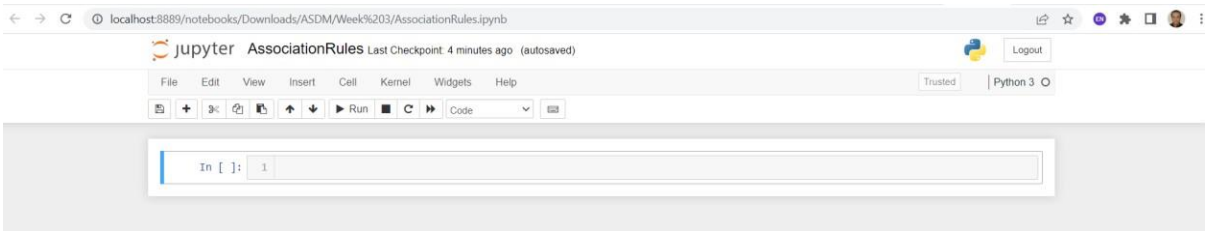
Part 1: Exercise

In this workshop we will practice how to implement Association Rule Mining using Python programming.

First, you need to run Anaconda Navigator



Then, launch Jupyter notebook among the applications lists.



Fortunately, we do not need to implement the apriori algorithm ourselves as there is already an implementation as part of the apyori package. This library is an implementation of Apriori Algorithm. You also need to download a python file namely ***ARutils.py*** from Blackboard. We have created this file to make inspecting and extracting the rules easier. **You will need to save this file in the same directory as the notebook you are working in.**

The dataset, ***marketbasket.csv***, that we will use in the workshop can be downloaded from Blackboard. We will be using market basket data to find purchasing behaviors of customers using association rule mining. We want to know that what items are bought together using association rules.

Data Explanation

Item	Purchase
Apples	Yes=Purchased, No=Not purchased
banana	Yes=Purchased, No=Not purchased
Coke	Yes=Purchased, No=Not purchased
Turkey	Yes=Purchased, No=Not purchased
bourbon	Yes=Purchased, No=Not purchased
ice-cream	Yes=Purchased, No=Not purchased
Baguette	Yes=Purchased, No=Not purchased
Soda	Yes=Purchased, No=Not purchased
Chocolate	Yes=Purchased, No=Not purchased
Cracker	Yes=Purchased, No=Not purchased
Cosmetics	Yes=Purchased, No=Not purchased
Avocado	Yes=Purchased, No=Not purchased
Artichoke	Yes=Purchased, No=Not purchased
Sardines	Yes=Purchased, No=Not purchased

Implementation

After installing the required libraries and running **Jupyter** notebook you can conduct the following steps:

1. Run the below pip install command to install the **apyori** package

```
In [1]: !pip install apyori
        Requirement already satisfie
```

2. Import all required libraries including **apyori** and ARutils. You **must** have saved the ARutils.py file in the same directory as your notebook before running this cell.

```
# Import required libraries
import matplotlib.pyplot as plt
import pandas as pd
from apyori import apriori
import ARutils
%matplotlib inline
```

3. Download ***marketbasket.csv*** dataset from Blackboard and save next to your Jupyter notebook.

4. Run the following code to read the file:

```
In [32]:
1 # Load the dataset
2 store_data = pd.read_csv('marketbasket.csv')
```

5. Inspect the dataset:

   Once the file has been imported to a pandas data frame, there are several built in methods to inspect it. **Dataframe.head()** displays the top rows. **Dataframe.tail()** shows the bottom rows and **Dataframe.describe()** describes the structure of a data frame.

```
In [34]:
1 # Analyze the dataset
2 store_data.head()
```

Out[34]:

	apples	banana	coke	turkey	bourbon	ice_cream	baguette	soda	choclate	cracker	cosmetics	avocado	artichoke
0	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No	No
1	No	No	Yes	No	Yes	No	Yes	Yes	No	No	Yes	Yes	No
2	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
3	No	No	Yes	Yes	Yes	No	Yes	No	No	No	Yes	No	No
4	No	Yes	No	No	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes

```
In [35]:
1 # Analyze the dataset
2 store_data.tail()
```

Out[35]:

	apples	banana	coke	turkey	bourbon	ice_cream	baguette	soda	choclate	cracker	cosmetics	avocado	artichoke
995	No	No	No	No	No	No	No	No	No	No	No	No	No
996	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No
997	No	Yes	Yes	Yes	Yes	No	No	Yes	No	No	Yes	No	Yes
998	Yes	Yes	No	No	Yes	No	No	No	Yes	Yes	No	No	Yes
999	No	No	No	No	Yes	No	Yes	Yes	No	Yes	No	No	No

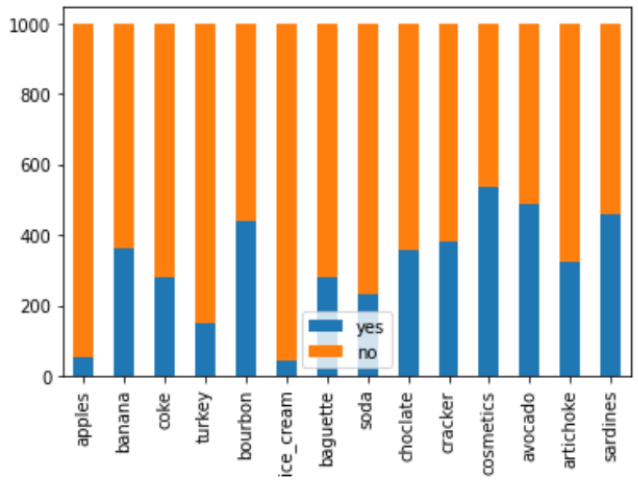
In [36]:
```
# Describe the dataset
store_data.describe()
```

Out[36]:

	apples	banana	coke	turkey	bourbon	ice_cream	baguette	soda	choclate	cracker	cosmetics	avocado	artichoke	sardines
count	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
unique	2	2	2	2	2	2	2	2	2	2	2	2	2	2
top	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No
freq	946	637	720	851	558	958	721	766	643	619	536	510	678	543

6. Plot and explore the "store_data" dataset with **Dataframe.plot.bar()** function to show how many values are in each columns and what their distributions are.

In [37]:
```
# Plot and explore the dataset
yes = (store_data=='Yes').sum()
no = (store_data=='No').sum()
purchased  = pd.concat([yes,no],axis=1,keys=['yes','no'])
ax = purchased.plot.bar(stacked=True)
plt.show()
```



Set stacked to False in the code above and observe the difference.

A simple example

7. The following example, containing 4 simple transactions, shows how the **apriori** algorithm in Python works.

In [23]:
```
"""
A small smaple for Apriori Algorithm in Python
"""
# Creating Sample Transactions
transactions = [
     ['Milk', 'Bread', 'Saffron'],
     ['Milk', 'Saffron'],
     ['Bread', 'Saffron','Wafer'],
     ['Bread','Wafer'],
        ]
transactions
```

Out[23]:
```
[['Milk', 'Bread', 'Saffron'],
 ['Milk', 'Saffron'],
 ['Bread', 'Saffron', 'Wafer'],
 ['Bread', 'Wafer']]
```

8. By passing a list of transactions (a list of lists), the minimum support, and the minimum confidence into the **apriori** class, a list of association rules will be obtained.

```
In [24]:
1 # Generating association rules
2 Rules = list(apriori(transactions, min_support=0.2, min_confidence=0.5))
```

9. **ARutils.extract** function, extracts the related rules as well as their support, confidence and lift. We can use this function to return the set of rules as a list of lists, which we can then use to create a pandas Data Frame.

```
# Extract the created rules
associationRules = ARutils.extract(Rules)
rules_df = pd.DataFrame(associationRules,columns=['LHS','RHS','Support','Confidence','Lift'])
rules_df
```

	LHS	RHS	Support	Confidence	Lift
0	[]	[Bread]	0.75	0.750000	1.000000
1	[]	[Milk]	0.50	0.500000	1.000000
2	[]	[Saffron]	0.75	0.750000	1.000000
3	[]	[Wafer]	0.50	0.500000	1.000000
4	[Milk]	[Bread]	0.25	0.500000	0.666667
5	[]	[Bread, Saffron]	0.50	0.500000	1.000000
6	[Bread]	[Saffron]	0.50	0.666667	0.888889
7	[Saffron]	[Bread]	0.50	0.666667	0.888889
8	[]	[Bread, Wafer]	0.50	0.500000	1.000000
9	[Bread]	[Wafer]	0.50	0.666667	1.333333
10	[Wafer]	[Bread]	0.50	1.000000	1.333333
11	[]	[Milk, Saffron]	0.50	0.500000	1.000000
12	[Milk]	[Saffron]	0.50	1.000000	1.333333

10. Refer back to the equations for support, confidence and lift provided earlier in this document, and the four simple transactions we created above, to check you're comfortable with how the support, confidence and lift values have been calculated for each rule. Where the LHS (the antecedent) is blank, the itemset is

the item(s) on the RHS alone. In these cases, the support and the confidence are the same – i.e., this is telling you how confident you can be that *any* basket will include the itemset on the RHS.

11. You can also view the rules generated using the **ARutils.inspect** function

```
ARutils.inspect(associationRules)
```

```
The number of associated rules: 23
LHS: [] --> RHS:['Bread'], support: 0.75, confidence: 0.75, lift: 1.00
--------------------------------------------------------------------------------
LHS: [] --> RHS:['Milk'], support: 0.50, confidence: 0.50, lift: 1.00
--------------------------------------------------------------------------------
LHS: [] --> RHS:['Saffron'], support: 0.75, confidence: 0.75, lift: 1.00
--------------------------------------------------------------------------------
LHS: [] --> RHS:['Wafer'], support: 0.50, confidence: 0.50, lift: 1.00
--------------------------------------------------------------------------------
LHS: ['Milk'] --> RHS:['Bread'], support: 0.25, confidence: 0.50, lift: 0.67
--------------------------------------------------------------------------------
LHS: [] --> RHS:['Bread', 'Saffron'], support: 0.50, confidence: 0.50, lift: 1.00
--------------------------------------------------------------------------------
LHS: ['Bread'] --> RHS:['Saffron'], support: 0.50, confidence: 0.67, lift: 0.89
--------------------------------------------------------------------------------
LHS: ['Saffron'] --> RHS:['Bread'], support: 0.50, confidence: 0.67, lift: 0.89
--------------------------------------------------------------------------------
LHS: [] --> RHS:['Wafer', 'Bread'], support: 0.50, confidence: 0.50, lift: 1.00
--------------------------------------------------------------------------------
LHS: ['Bread'] --> RHS:['Wafer'], support: 0.50, confidence: 0.67, lift: 1.33
--------------------------------------------------------------------------------
LHS: ['Wafer'] --> RHS:['Bread'], support: 0.50, confidence: 1.00, lift: 1.33
--------------------------------------------------------------------------------
LHS: [] --> RHS:['Milk', 'Saffron'], support: 0.50, confidence: 0.50, lift: 1.00
```

12. Data Pre-processing: by taking a brief look at the above example, we can find out that **apriori** library requires a dataset to be as a list of lists in which the entire dataset is a big list of each transaction; whilst our data is pandas data frame. To convert this data frame into a list of lists, **ARutils.data_prepare** has been implemented.

```
# Pre process the main dataset
transactions = ARutils.data_prepare(store_data)
```

13. Applying Apriori Algorithm; the next step is to apply the Apriori algorithm on the dataset. To do so, we can use the **apriori** class that we imported from the apyori library. This class requires some parameter values to work. The first parameter is the list of lists that you want to extract rules from. The second parameter is the **min_support** parameter. This parameter is used to select the

items with support values greater than the value specified by the parameter. Next, the **min_confidence** parameter return those rules that have confidence greater than the confidence threshold specified by the parameter.

Using the **apriori** class by sending the transactions object and identifying the minimum support and confidence you can extract and inspect rules on the ***marketbasket*** dataset. As you can see, there are 27,348 rules in total!

```
In [18]: # Create association rules on the main dataset
         Rules = list(apriori(transactions, min_support=0.02, min_confidence=0.2))
         associationRules = ARutils.extract(Rules)
         rules_df = pd.DataFrame(associationRules,columns=['LHS','RHS','Support','Confidence','Lift'])
         len(rules_df)

Out[18]: 27348
```

14. One of the advantages of using pandas Data Frames is that we can easily filter and sort the data within it. For example, let's take a look at the top ten rules with the highest lift. To do that, we can use the nlargest() method.

```
In [18]: rules_df.nlargest(10, "Lift")
```

Out[18]:

	LHS	RHS	Support	Confidence	Lift
26986	[choclate, bourbon, baguette, coke]	[banana, turkey, soda, cracker]	0.022	0.400000	16.666667
27024	[banana, turkey, soda, cracker]	[choclate, bourbon, baguette, coke]	0.022	0.916667	16.666667
26990	[bourbon, baguette, coke, cracker]	[choclate, banana, turkey, soda]	0.022	0.423077	16.272189
27020	[choclate, banana, turkey, soda]	[bourbon, baguette, coke, cracker]	0.022	0.846154	16.272189
27212	[sardines, baguette, coke, cracker]	[choclate, bourbon, turkey, soda]	0.021	0.552632	16.253870
27230	[choclate, bourbon, turkey, soda]	[sardines, baguette, coke, cracker]	0.021	0.617647	16.253870
27013	[banana, bourbon, turkey, cracker]	[choclate, coke, baguette, soda]	0.022	0.647059	16.176471
26997	[choclate, soda, baguette, coke]	[banana, bourbon, turkey, cracker]	0.022	0.550000	16.176471
27176	[choclate, turkey, soda]	[baguette, coke, cracker, sardines, bourbon]	0.021	0.552632	15.789474
27266	[baguette, coke, cracker, sardines, bourbon]	[choclate, turkey, soda]	0.021	0.600000	15.789474

15. We can use the same method to identify the 10 rules with the highest support and confidence too:

```
In [19]: rules_df.nlargest(10, "Support")
```

Out[19]:

	LHS	RHS	Support	Confidence	Lift
7	[]	[cosmetics]	0.536	0.536000	1.000000
1	[]	[avocado]	0.490	0.490000	1.000000
9	[]	[sardines]	0.457	0.457000	1.000000
4	[]	[bourbon]	0.442	0.442000	1.000000
8	[]	[cracker]	0.381	0.381000	1.000000
3	[]	[banana]	0.363	0.363000	1.000000
5	[]	[choclate]	0.357	0.357000	1.000000
55	[]	[avocado, cosmetics]	0.356	0.356000	1.000000
56	[avocado]	[cosmetics]	0.356	0.726531	1.355468
57	[cosmetics]	[avocado]	0.356	0.664179	1.355468

```
In [20]: rules_df.nlargest(10, "Confidence")
```

Out[20]:

	LHS	RHS	Support	Confidence	Lift
144	[turkey]	[coke]	0.149	1.0	3.571429
209	[apples, turkey]	[coke]	0.020	1.0	3.571429
363	[turkey, artichoke]	[coke]	0.049	1.0	3.571429
551	[avocado, turkey]	[coke]	0.068	1.0	3.571429
726	[turkey, baguette]	[coke]	0.097	1.0	3.571429
860	[banana, turkey]	[coke]	0.070	1.0	3.571429
965	[bourbon, turkey]	[coke]	0.092	1.0	3.571429
1038	[choclate, turkey]	[coke]	0.083	1.0	3.571429
1058	[choclate, ice_cream]	[cracker]	0.023	1.0	2.624672
1104	[turkey, cosmetics]	[coke]	0.074	1.0	3.571429

16. We can see that the top rules that have highest support are those with an empty itemset on the LHS. However, these aren't particularly useful as rules as they don't provide any association. We can use a lambda function to filter out those rules which have an empty list in the LHS column of the Data Frame.

```
In [47]: rules_df[rules_df['LHS'].apply(lambda x: len(x) > 0)].nlargest(10, "Support")
```

Out[47]:

	LHS	RHS	Support	Confidence	Lift
56	[avocado]	[cosmetics]	0.356	0.726531	1.355468
57	[cosmetics]	[avocado]	0.356	0.664179	1.355468
126	[choclate]	[cracker]	0.321	0.899160	2.359999
127	[cracker]	[choclate]	0.321	0.842520	2.359999
115	[bourbon]	[sardines]	0.297	0.671946	1.470341
116	[sardines]	[bourbon]	0.297	0.649891	1.470341
149	[cosmetics]	[sardines]	0.238	0.444030	0.971619
150	[sardines]	[cosmetics]	0.238	0.520788	0.971619
108	[bourbon]	[cosmetics]	0.231	0.522624	0.975046
109	[cosmetics]	[bourbon]	0.231	0.430970	0.975046

17. One way we can reduce the number of rules is to decrease the size of the maximum itemset. (For example, with 14 items, there are 182 potential two-item permutations, 2,184 potential three-item permutations and 24,024 potential four-item permutations – setting a maximum itemset size will therefore help keep the number of rules to a more manageable number.) Lets run this again with a maximum itemset of three.

```
In [17]: # Create association rules on the main dataset
         Rules = list(apriori(transactions, min_support=0.02, min_confidence=0.2, max_length=3))
         associationRules = ARutils.extract(Rules)
         rules_df = pd.DataFrame(associationRules,columns=['LHS','RHS','Support','Confidence','Lift'])
         len(rules_df)

Out[17]: 1168
```

18. We have now got a more manageable list of rules which we can explore as above

```
In [24]: rules_df.nlargest(10, "Lift")
```

Out[24]:

	LHS	RHS	Support	Confidence	Lift
724	[turkey]	[baguette, coke]	0.097	0.651007	5.208054
725	[baguette, coke]	[turkey]	0.097	0.776000	5.208054
1130	[turkey]	[coke, soda]	0.068	0.456376	4.907267
1131	[soda, coke]	[turkey]	0.068	0.731183	4.907267
208	[apples, coke]	[turkey]	0.020	0.714286	4.793864
1036	[turkey]	[choclate, coke]	0.083	0.557047	4.157067
1037	[choclate, coke]	[turkey]	0.083	0.619403	4.157067
1116	[coke, cracker]	[turkey]	0.081	0.618321	4.149803
1115	[turkey]	[coke, cracker]	0.081	0.543624	4.149803
963	[turkey]	[bourbon, coke]	0.092	0.617450	4.143957

```
In [25]: rules_df.nlargest(10, "Support")
```

Out[25]:

	LHS	RHS	Support	Confidence	Lift
7	[]	[cosmetics]	0.536	0.536000	1.000000
1	[]	[avocado]	0.490	0.490000	1.000000
9	[]	[sardines]	0.457	0.457000	1.000000
4	[]	[bourbon]	0.442	0.442000	1.000000
8	[]	[cracker]	0.381	0.381000	1.000000
3	[]	[banana]	0.363	0.363000	1.000000
5	[]	[choclate]	0.357	0.357000	1.000000
55	[]	[avocado, cosmetics]	0.356	0.356000	1.000000
56	[avocado]	[cosmetics]	0.356	0.726531	1.355468
57	[cosmetics]	[avocado]	0.356	0.664179	1.355468

```
In [26]: rules_df.nlargest(10, "Confidence")
```

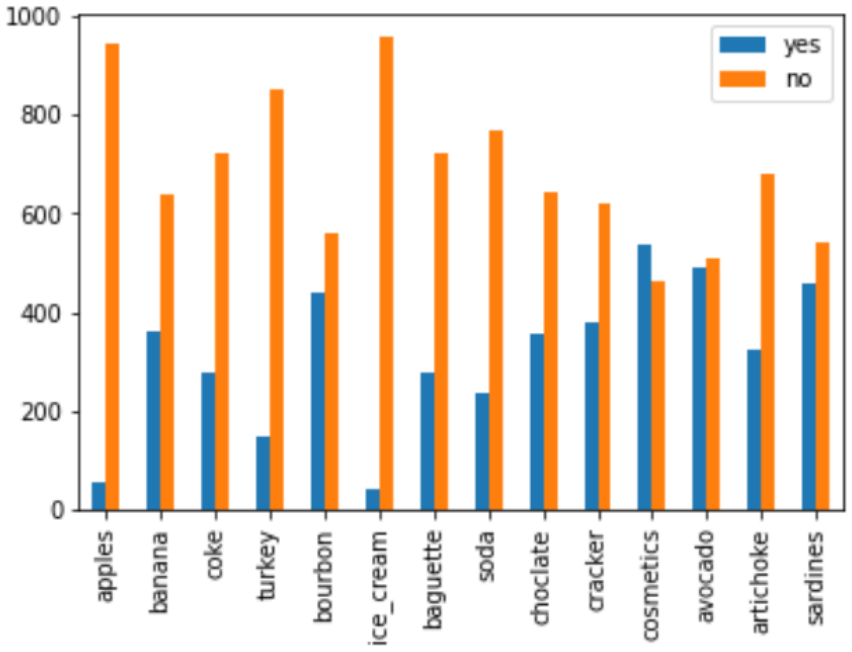
Out[26]:

	LHS	RHS	Support	Confidence	Lift
144	[turkey]	[coke]	0.149	1.0	3.571429
209	[apples, turkey]	[coke]	0.020	1.0	3.571429
363	[turkey, artichoke]	[coke]	0.049	1.0	3.571429
551	[avocado, turkey]	[coke]	0.068	1.0	3.571429
726	[turkey, baguette]	[coke]	0.097	1.0	3.571429
860	[banana, turkey]	[coke]	0.070	1.0	3.571429
965	[bourbon, turkey]	[coke]	0.092	1.0	3.571429
1038	[choclate, turkey]	[coke]	0.083	1.0	3.571429
1058	[choclate, ice_cream]	[cracker]	0.023	1.0	2.624672
1104	[turkey, cosmetics]	[coke]	0.074	1.0	3.571429

19. Since we are more interested what customers are purchasing, let's explore those items which are most popular and the rules associated with them. First of all, we should find the most popular products based on sales dataset. Plotting could be the easiest way to find the most purchased item.

In [29]:

```
# Investigate the dataset in more depth
ax = purchased.plot.bar()
plt.show()
```



20. According to the plot, cosmetics are the most popular items. Since we want to see rules where customers are buying more of the items, so use the following code to get those rules for cosmetics:

In [23]:

```
# Find the cosmetic rules
Rules = list(apriori(transactions, min_support=0.1, min_confidence=0.95))
associationRules = ARutils.extract(Rules,'cosmetics',2)
ARutils.inspect(associationRules)
```

```
The number of associated rules: 0
```

21. Regarding the result, there are no rules available for the given parameter (the minimum confidence is too high in this case). Change the confidence parameter value to 50% (0.50), A lower confidence threshold allows more rules to show up.

In [28]:
```
Rules = list(apriori(transactions, min_support=0.1, min_confidence=0.5))
associationRules = ARutils.extract(Rules,'cosmetics',2)
ARutils.inspect(associationRules)
```

```
The number of associated rules: 24
LHS: ['artichoke'] --> RHS:['cosmetics'], support: 0.17, confidence: 0.52, lift: 0.97
--------------------------------------------------------------------------------
LHS: ['avocado'] --> RHS:['cosmetics'], support: 0.36, confidence: 0.73, lift: 1.36
--------------------------------------------------------------------------------
LHS: ['banana'] --> RHS:['cosmetics'], support: 0.19, confidence: 0.53, lift: 0.99
--------------------------------------------------------------------------------
LHS: ['bourbon'] --> RHS:['cosmetics'], support: 0.23, confidence: 0.52, lift: 0.98
--------------------------------------------------------------------------------
LHS: ['choclate'] --> RHS:['cosmetics'], support: 0.19, confidence: 0.54, lift: 1.00
--------------------------------------------------------------------------------
LHS: ['coke'] --> RHS:['cosmetics'], support: 0.14, confidence: 0.51, lift: 0.95
--------------------------------------------------------------------------------
LHS: ['cracker'] --> RHS:['cosmetics'], support: 0.21, confidence: 0.55, lift: 1.03
--------------------------------------------------------------------------------
LHS: ['sardines'] --> RHS:['cosmetics'], support: 0.24, confidence: 0.52, lift: 0.97
--------------------------------------------------------------------------------
LHS: ['artichoke', 'avocado'] --> RHS:['cosmetics'], support: 0.12, confidence: 0.73, lift: 1.37
--------------------------------------------------------------------------------
LHS: ['avocado', 'banana'] --> RHS:['cosmetics'], support: 0.12, confidence: 0.67, lift: 1.25
--------------------------------------------------------------------------------
LHS: ['avocado', 'bourbon'] --> RHS:['cosmetics'], support: 0.14, confidence: 0.65, lift: 1.21
--------------------------------------------------------------------------------
```

Remark: The level of confidence and support which is appropriate to use will depend on your dataset; the number of transactions, number of items, and the average number of times an item shows up in transaction. You will likely need to experiment with different values of confidence and support to get a manageable number of useful rules.

22. We can also plot these rules interactively using plotly express. This allows us to create interactive plots quickly and easily. You will first need to use the pip install command to install the required packages.

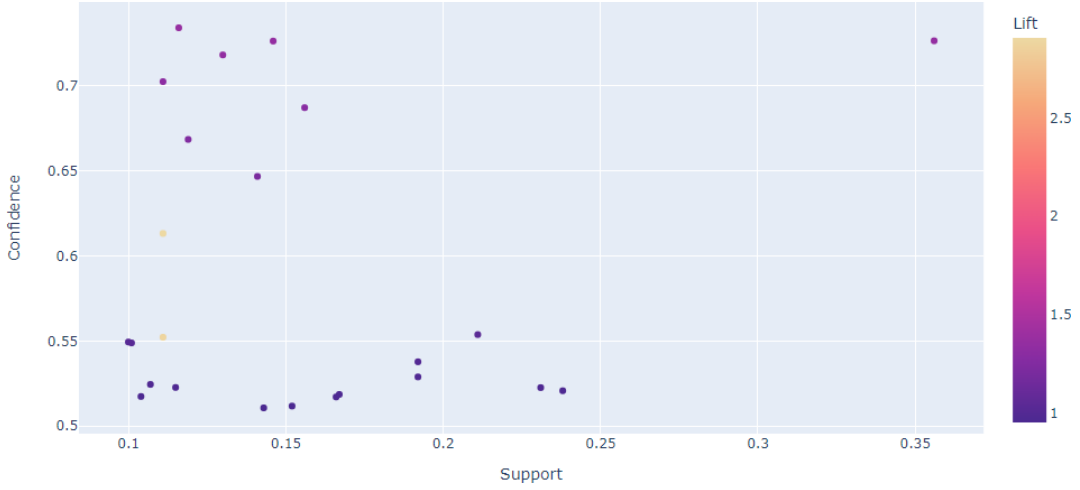
In [72]:
```
!pip install plotly==5.10.0
```

23. Run the code below, and you can then hover the mouse over any of the points on the plot to view the details of the rule.

```
In [39]: rules_df = pd.DataFrame(associationRules,columns=['LHS','RHS','Support','Confidence','Lift'])

         import plotly.express as px

         fig = px.scatter(rules_df, x="Support", y="Confidence", color="Lift",
                          hover_data=['LHS','RHS'], color_continuous_scale="agsunset")
         fig.show()
```



24. As well as using **ARutils.extract** to apply a filter, you can also generate the rules and apply a filter to the pandas Data Frame itself. To do this we use a lambda function to perform a partial string match. If you're not familiar with lambda functions, you can find out more here.

In [27]:
```
Rules = list(apriori(transactions, min_support=0.02, min_confidence=0.2, max_length=3))
associationRules = ARutils.extract(Rules)
rules_df = pd.DataFrame(associationRules,columns=['LHS','RHS','Support','Confidence','Lift'])#

#Use lambda function for partial string match

rules_df[rules_df['RHS'].apply(lambda x: 'avocado' in x)]
```

Out[27]:

	LHS	RHS	Support	Confidence	Lift
1	[]	[avocado]	0.490	0.490000	1.000000
11	[apples]	[avocado]	0.030	0.555556	1.133787
22	[artichoke]	[avocado]	0.158	0.490683	1.001394
45	[baguette]	[avocado]	0.141	0.505376	1.031380
47	[banana]	[avocado]	0.178	0.490358	1.000731
...	...	...	...	...	...
584	[sardines, soda]	[avocado]	0.063	0.484615	0.989011
585	[turkey]	[sardines, avocado]	0.037	0.248322	1.093930
587	[sardines, turkey]	[avocado]	0.037	0.474359	0.968080
588	[turkey]	[avocado, soda]	0.030	0.201342	1.735709

25. It's also useful to know how to sort a pandas Data Frame. We can do this by using the sort_values() method.

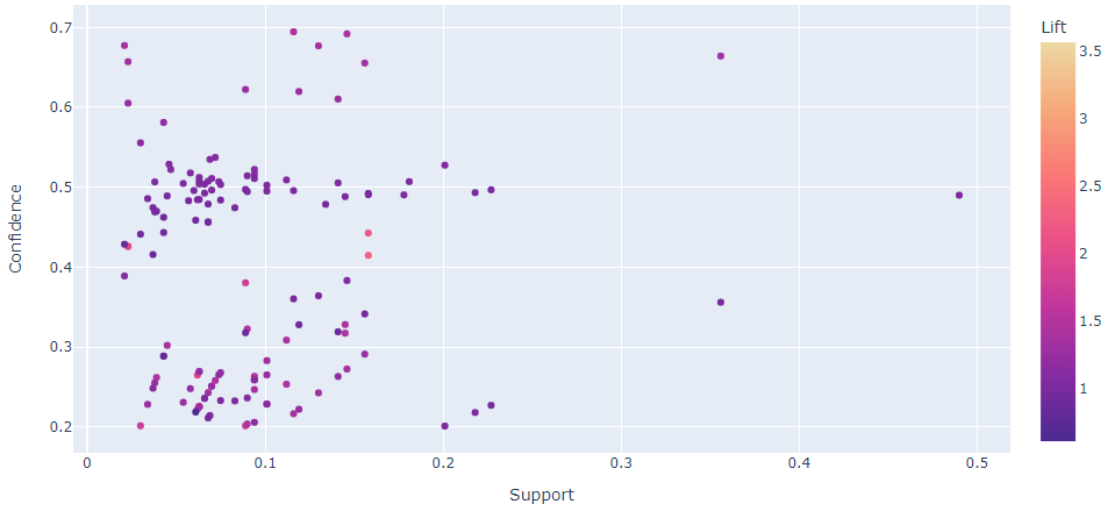
In [41]:
```
avocado_rules = rules_df[rules_df['RHS'].apply(lambda x: 'avocado' in x)].sort_values(by=['Lift'], ascending=False)
avocado_rules.head()
```

Out[41]:

	LHS	RHS	Support	Confidence	Lift
548	[coke]	[avocado, turkey]	0.068	0.242857	3.571429
549	[turkey]	[avocado, coke]	0.068	0.456376	3.405790
168	[apples]	[avocado, choclate]	0.023	0.425926	2.353182
517	[cracker]	[avocado, choclate]	0.158	0.414698	2.291150
516	[choclate]	[avocado, cracker]	0.158	0.442577	2.201876

26. We can explore these rules using another interactive scatterplot, as below:

In [50]:
```
fig = px.scatter(avocado_rules, x="Support", y="Confidence", color="Lift",
                 hover_data=['LHS','RHS'], color_continuous_scale="agsunset")
fig.show()
```



When analysing rules, a common approach is to divide up the rules into three categories:

- Actionable
- Trivial
- Inexplicable

Trivial rules are those which are obvious and don't provide any new insights. For example, *diapers* ➔ *baby formula* falls into this category. The inexplicable ones are those that are difficult to understand and therefore don't provide any actionable insights – for example, perhaps there is just one customer who is purchasing the same two items together regularly! The aim is to identify the actionable rules – for example, *diapers* ➔ *beer* – which we might not have predicted but which give us something we can action. Understanding which rules fall into this category is where your domain knowledge as a Data Scientist comes in!

Part 2: Exercise

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this exercise, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the Association Rule mining to predict which passengers survived from the tragedy.

titanic_preprocessed.csv data set can be downloaded from the Blackboard. We have pre-processed the data for you so this is in the format required to use the ARutils.prepare_data() function. The columns are:

- **Age_Adult**
- **Age_Child**
- **Sex_Female**
- **Sex_Male**
- **Crew_Member**
- **First_class**
- **Second_class**
- **Third_class**
- **Survived**

Each variable takes the value 'Yes' or 'No'; the data takes the value 'Yes' when the attribute applies to the individual in the data.

For this exercise, you should apply Association Rule mining and then use a lambda function similar to the one used in step 24 to filter on those rules that have 'Survived' on the RHS of the rule.

source : https://www.kaggle.com/c/titanic