PHASE5:Development and Project Submission

Project title:Market Basket Insights using Python

Welcome to this hands-on training event on Market Basket Analysis in Python. In this session, you will learn how to:

- Identify patterns in consumer decision-making with the mlxtend package.
- Use metrics to evaluate the properties of patterns.
- Construct "rules" that provide concrete recommendations for businesses.
- Visualize patterns and rules using seaborn and matplotlib.

The dataset

We'll use a dataset from a Brazilian ecommerce site (olist.com) that is divided into three CSV files:

1. grocery.csv

The column definitions are as follows:

olist_order_items_dataset.csv:

- order_id: The unique identifier for a transaction.
- order_item_id: The order of an item within a transaction.
- product_id: The unique identifier for a product.
- price: The product's price.

olist_products_dataset.csv:

- product_id: The unique identifier for a product.
- product_category_name: The name of an item's product category in Portuguese.
- product_weight_g: The product's weight in grams.
- product_length_cm: The product's length in centimeters.
- product_width_cm: The product's width in centimeters.
- product_height_cm: The product's height in centimeters.

product_category_name_translation.csv:

- product_category_name: The name of an item's product category in Portuguese.
- product_category_name_english: The name of an item's product category in English.

Data preparation

The first step in any Market Basket Analysis (MBA) project is to determine what constitutes an **item**, an **itemset**, and a **transaction**. This will depend on the dataset we're using and the question we're attempting to answer.

- **Grocery store**
  - Item: Grocery
  - Itemset: Collection of groceries
  - Transaction: Basket of items purchased
- **Music streaming service**

    - Item: Song
    - Itemset: Collection of unique songs
    - Transaction: User song library
- **Ebook store**
    - Item: Ebook
    - Itemset: One or more ebooks
    - Transaction: User ebook library

In this live training session, we'll use a dataset of transactions from olist.com, a Brazilian ecommerce site.

- 100,000+ orders over 2016-2018.
- Olist connects sellers to marketplaces.
- Seller can register products with Olist.
- Customer makes purchase at marketplace from Olist store.
- Seller fulfills orders.

What is an item?

- A product purchased from Olist.

What is an itemset?

- A collection of one or more product(s).

What is a transaction?

- An itemset that corresponds to a customer's order.

In [1]:

```
# Import modules.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Set default asthetic parameters.
sns.set()

# Define path to data.
data_path = 'https://github.com/datacamp/Market-Basket-Analysis-in-python-live-training/raw/master/data/'
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [2]:

```
# Load orders dataset.
orders = pd.read_csv(data_path+'grocery.csv')
```

```python
# Load products items dataset.
products = pd.read_csv(data_path+'olist_products_dataset.csv')

# Load translations dataset.
translations =
pd.read_csv(data_path+'product_category_name_translation.csv')
```

In [3]:

```python
# Print orders header.
orders.head()
```

Out[3]:

	order_id	order_item_id	product_id	price
0	b8bfa12431142333a0c84802f9529d87	1	765a8070ece0f1383d0f5faf913dfb9b	81.0
1	b8bfa12431142333a0c84802f9529d87	2	a41e356c76fab66334f36de622ecbd3a	99.3
2	b8bfa12431142333a0c84802f9529d87	3	765a8070ece0f1383d0f5faf913dfb9b	81.0
3	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	58.9
4	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	239.9

In [4]:

```python
# Print orders info.
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112650 entries, 0 to 112649
Data columns (total 4 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   order_id       112650 non-null  object
 1   order_item_id  112650 non-null  int64
 2   product_id     112650 non-null  object
 3   price          112650 non-null  float64
dtypes: float64(1), int64(1), object(2)
memory usage: 3.4+ MB
```

In [5]:

```python
# Print products header.
products.head()
```

Out[5]:

	product_id	product_category_name	product_weight_g	product_length_cm	product_height_cm	product_width_cm
0	1e9e8ef04dbcff4541ed26657ea517e5	perfumaria	225.0	16.0	10.0	14.0
1	3aa071139cb16b67ca9e5dea641aaa2f	artes	1000.0	30.0	18.0	20.0
2	96bd76ec8810374ed1b65e291975717f	esporte_lazer	154.0	18.0	9.0	15.0
3	cef67bcfe19066a932b7673e239eb23d	bebes	371.0	26.0	4.0	26.0
4	9dc1a7de274444849c219cff195d0b71	utilidades_domesticas	625.0	20.0	17.0	13.0

In [6]:

```
# Print products info.
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32951 entries, 0 to 32950
Data columns (total 6 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   product_id             32951 non-null  object
 1   product_category_name  32341 non-null  object
 2   product_weight_g       32949 non-null  float64
 3   product_length_cm      32949 non-null  float64
 4   product_height_cm      32949 non-null  float64
 5   product_width_cm       32949 non-null  float64
dtypes: float64(4), object(2)
memory usage: 1.5+ MB
```

In [7]:

```
# Print translations header.
translations.head()
```

Out[7]:

	product_category_name	product_category_name_english
0	beleza_saude	health_beauty
1	informatica_acessorios	computers_accessories

	product_category_name	product_category_name_english
2	automotivo	auto
3	cama_mesa_banho	bed_bath_table
4	moveis_decoracao	furniture_decor

In [8]:

```
# Print translations info.
translations.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71 entries, 0 to 70
Data columns (total 2 columns):
 #   Column                         Non-Null Count  Dtype 
---  ------                         --------------  ----- 
 0   product_category_name          71 non-null     object
 1   product_category_name_english  71 non-null     object
dtypes: object(2)
memory usage: 1.2+ KB
```

Q&A 1

Translating item category names

The product names are given in Portuguese.

- We'll translate the names to English using a pandas DataFrame named translations.
- .merge() performs a join operation on columns or indices.
- on is the column on which to perform the join.
- how specifies which keys to use to perform the join.

In [9]:

```
# Translate product names to English.
products = products.merge(translations, on='product_category_name',
how="left")

# Print English names.
products['product_category_name_english']
```

Out[9]:

```
0                         perfume
1                             art
2                 sports_leisure
3                            baby
4                      housewares
```

```
                 ...
32946              furniture_decor
32947    construction_tools_lights
32948               bed_bath_table
32949        computers_accessories
32950               bed_bath_table
Name: product_category_name_english, Length: 32951, dtype: object
```

Convert product IDs to product category names.

We can work with product IDs directly, but do not have product names.

- Map product IDs to product category names, which are available in products.
- Use another .merge() with orders and subset of products columns.

Using category names will also simplify the analysis, since there are fewer categories than products.

In [10]:

```
# Define product category name in orders DataFrame.
orders =
orders.merge(products[['product_id','product_category_name_english']],
on='product_id', how='left')
```

In [11]:

```
# Print orders header.
orders.head()
```

Out[11]:

	order_id	order_item_id	product_id	price	product_category_name_english
0	b8bfa12431142333a0c84802f9529d87	1	765a8070ece0f1383d0f5faf913dfb9b	81.0	sports_leisure
1	b8bfa12431142333a0c84802f9529d87	2	a41e356c76fab66334f36de622ecbd3a	99.3	NaN
2	b8bfa12431142333a0c84802f9529d87	3	765a8070ece0f1383d0f5faf913dfb9b	81.0	sports_leisure
3	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	58.9	cool_stuff
4	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	239.9	pet_shop

In [12]:

```
# Drop products without a defined category.
orders.dropna(inplace=True, subset=['product_category_name_english'])
```

In [13]:

```
# Print number of unique items.
len(orders['product_id'].unique())
```

Out[13]:

```
32328
```

In [14]:

```
# Print number of unique categories.
len(orders['product_category_name_english'].unique())
```

Out[14]:

```
71
```

Insight: Performing "aggregation" up to the product category level reduces the number of potential itemsets from 232328 to 271.

Construct transactions from order and product data

- **We will perform Market Basket Analysis on transactions.**
  - A transaction consists of the unique items purchased by a customer.
- **Need to extract transactions from orders DataFrame.**
  - Group all items in an order.

In [15]:

```
# Identify transactions associated with example order.
example1 = orders[orders['order_id'] ==
'fe64170e936bc5f6a6a41def260984b9']['product_category_name_english']

# Print example.
example1
```

Out[15]:

```
111984     bed_bath_table
111985    furniture_decor
Name: product_category_name_english, dtype: object
```

In [16]:

```
# Identify transactions associated with example order.
example2 = orders[orders['order_id'] ==
'fffb9224b6fc7c43ebb0904318b10b5f']['product_category_name_english']

# Print example.
example2
```

Out[16]:

```
112640    watches_gifts
112641    watches_gifts
112642    watches_gifts
112643    watches_gifts
Name: product_category_name_english, dtype: object
```

Insight: Aggregation reduces the number of items and, therefore, itemsets.

Map orders to transactions.

- .groupby() splits a DataFrame into groups according to some criterion.
- .unique() returns list of unique values.

In [17]:

```
# Recover transaction itemsets from orders DataFrame.
```

```
transactions =
orders.groupby("order_id").product_category_name_english.unique()

# Print transactions header.
transactions.head()
```

Out[17]:

```
order_id
00010242fe8c5a6d1ba2dd792cb16214         [cool_stuff]
00018f77f2f0320c557190d7a144bdd3           [pet_shop]
000229ec398224ef6ca0657da4fc703e    [furniture_decor]
00024acbcdf0a6daa1e931b038114c75            [perfume]
00042b26cf59d7ce69dfabb4e55b4fd9       [garden_tools]
Name: product_category_name_english, dtype: object
```

In [18]:

```
# Plot 50 largest categories of transactions.
transactions.value_counts()[:50].plot(kind='bar', figsize=(15,5))
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff4bcf8b358>
```

Insight 1: The most common itemsets consist of a single item.

Insight 2: There's a long tail of categories that consist of infrequently purchased items.

Use .tolist() to transform a DataFrame or Series object into a list.

In [19]:

```
# Convert the pandas series to list of lists.
transactions = transactions.tolist()
```

Summarize final transaction data

In [20]:

```
# Print length of transactions.
len(transactions)
```

Out[20]:

```
97256
```

In [21]:

```
# Count number of unique item categories for each transaction.
counts = [len(transaction) for transaction in transactions]
```

In [22]:

```
# Print median number of items in a transaction.
np.median(counts)
```

Out[22]:

```
1.0
```

In [23]:

```
# Print maximum number of items in a transaction.
np.max(counts)
```

Out[23]:

```
3
```

Q&A 2

Association Rules and Metrics

Association rule: an "if-then" relationship between two itemsets.

- **rule:** if *{coffee)* then *{milk}*.
- **antecedent:** coffee
- **consequent:** milk

Metric: a measure of the strength of association between two itemsets.

- **rule:** if *{coffee)* then *{milk}*
- **support:** 0.10
- **leverage:** 0.03

One-hot encode the transaction data

- **One-hot encoding data.**
  - TransactionEncoder() instantiates an encoder object.
  - .fit() creates mapping between list and one-hot encoding.
  - .transform() transforms list into one-hot encoded array.

- **Applying one-hot encoding will transform the list of lists (of transactions) into a DataFrame.**
  - The columns correspond to item categories and the rows correspond to transactions. A true indicates that a transaction contains an item from the corresponding category.
- **One-hot encoding simplifies the computation of metrics.**
  - We will also use a one-hot encoded DataFrame as an input to different mlxtend functions.

In [24]:

```
from mlxtend.preprocessing import TransactionEncoder

# Instantiate an encoder.
encoder = TransactionEncoder()

# Fit encoder to list of lists.
encoder.fit(transactions)

# Transform lists into one-hot encoded array.
onehot = encoder.transform(transactions)

# Convert array to pandas DataFrame.
onehot = pd.DataFrame(onehot, columns = encoder.columns_)
```

In [25]:

```
# Print header.
onehot.head()
```

Out[25]:

Compute the support metric

Support measures the frequency with which an itemset appears in a database of transactions.

�������(�)=number of transactions containing Xtotal number of transactions

- .mean(axis=0) computes support values for one-hot encoded DataFrame.
- A high support value indicates that items in an itemset are purchased together frequently and, thus, are associated with each other.

In [26]:

```
# Print support metric over all rows for each column.
onehot.mean(axis=0)
```

Out[26]:

```
agro_industry_and_commerce    0.001871
air_conditioning              0.002601
art                           0.002077
arts_and_crafts               0.000236
audio                         0.003599
                                ...
stationery                    0.023762
tablets_printing_image        0.000812
telephony                     0.043175
toys                          0.039956
watches_gifts                 0.057827
Length: 71, dtype: float64
```

Observation: In retail and ecommerce settings, any particular item is likely to account for a small share of transactions. Here, we've aggregated up to the product category level and very popular categories are still only present in 5% of transactions. Consequently, itemsets with 2 or more item categories will account for a vanishingly small share of total transactions (e.g. 0.01%).

Compute the item count distribution over transactions

- onehot.sum(axis=1) sums across the columns in a DataFrame.

In [27]:

```
# Print distribution of item counts.
onehot.sum(axis=1).value_counts()
```

Out[27]:

```
1    96530
2      711
3       15
dtype: int64
```

Insight: Only 726 transactions contain more than one item category. We may want to consider whether aggregation discards too many multi-item itemsets.

Create a column for an itemset with multiple items

- **We can create multi-item columns using the logical AND operation.**
  - True & True = True
  - True & False = False
  - False & True = False
  - False & False = False

In [28]:

```
# Add sports_leisure and health_beauty to DataFrame.
```

```
onehot['sports_leisure_health_beauty'] = onehot['sports_leisure'] &
onehot['health_beauty']

# Print support value.
onehot['sports_leisure_health_beauty'].mean(axis = 0)
```

Out[28]:

```
0.00014394998766142962
```

Insight: Only 0.014% of transactions contain a product from both the sports and leisure, and health and beauty categories. These are typically the type of numbers we will work with when we set pruning thresholds in the following section.

Aggregate the dataset further by combining product sub-categories

- **We can use the inclusive OR operation to combine multiple categories.**
  - True | True = True
  - True | False = True
  - False | True = True
  - False | False = False

In [29]:

```
# Merge books_imported and books_technical.
onehot['books'] = onehot['books_imported'] | onehot['books_technical']

# Print support values for books, books_imported, and books_technical.
onehot[['books','books_imported','books_technical']].mean(axis=0)
```

Out[29]:

```
books              0.003218
books_imported     0.000545
books_technical    0.002673
dtype: float64
```

Compute the confidence metric

- **The support metric doesn't provide information about direction.**
  - �������(����������,����������)=�������(����������,����������)
- **The confidence metric has a direction.**
  - Conditional probability of the consequent, given the antecedent.

����������(����������→����������)=�������(����������,����������)�������(����������)

- A high value of confidence indicates that the antecedent and consequent are associated and that the direction of the association runs from the antecedent to the consequent.

In [30]:

```
# Compute joint support for sports_leisure and health_beauty.
joint_support = (onehot['sports_leisure'] & onehot['health_beauty']).mean()

# Print confidence metric for sports_leisure -> health_beauty.
joint_support / onehot['sports_leisure'].mean()
```

Out[30]:

```
0.0018134715025906734
```

In [31]:

```
# Print confidence for health_beauty -> sports_leisure.
joint_support / onehot['sports_leisure'].mean()
```

Out[31]:

```
0.0018134715025906734
```

Insight: ����������(������_�������→*h*����*h*_������) was higher than ����������(*h*����*h*_������→������_�������). Since the two have the same joint support, the confidence measures will differ only by the antecedent support. The higher confidence metric means that the antecedent has *lower* support.

Q&A 3

The Apriori Algorithm and Pruning

The Apriori algorithm identifies frequent (high support) itemsets using something called the Apriori principle, which states that a superset that contains an infrequent item is also infrequent.

Pruning is the process of removing itemsets or association rules, typically based on the application of a metric threshold.

The mlxtend module will enable us to apply the Apriori algorithm, perform pruning, and compute association rules.

Applying the Apriori algorithm

- Use apriori() to identify frequent itemsets.
- min_support set the item frequency threshold used for pruning.

In [32]:

```
from mlxtend.frequent_patterns import apriori

# Apply apriori algorithm to data with min support threshold of 0.01.
frequent_itemsets = apriori(onehot, min_support = 0.01)

# Print frequent itemsets.
frequent_itemsets
```

Out[32]:

	support	itemsets
0	0.040070	(5)

	support	itemsets
1	0.029664	(6)
2	0.096827	(7)
3	0.068777	(15)
4	0.010920	(16)
5	0.037345	(20)
6	0.026219	(27)
7	0.019166	(28)
8	0.066310	(40)
9	0.036173	(43)
10	0.090853	(44)
11	0.060500	(50)
12	0.010632	(53)
13	0.013089	(57)
14	0.032512	(59)
15	0.017582	(60)
16	0.079378	(65)
17	0.023762	(66)

	support	itemsets
18	0.043175	(68)
19	0.039956	(69)
20	0.057827	(70)

Observation 1: apriori returns a DataFrame with a support column and an itemsets column.

Observation 2: By default apriori returns itemset numbers, rather than labels. We can change this by using the use_colnames parameter.

Insight: All itemsets with a support of greater than 0.01 contain a single item.

- Use use_colnames to use item names, rather than integer IDs.

In [33]:

```
# Apply apriori algorithm to data with min support threshold of 0.001.
frequent_itemsets = apriori(onehot, min_support = 0.001, use_colnames =
True)

# Print frequent itemsets.
frequent_itemsets
```

Out[33]:

	support	itemsets
0	0.001871	(agro_industry_and_commerce)
1	0.002601	(air_conditioning)
2	0.002077	(art)
3	0.003599	(audio)
4	0.040070	(auto)
5	0.029664	(baby)
6	0.096827	(bed_bath_table)

	support	itemsets
7	0.005264	(books_general_interest)
8	0.002673	(books_technical)
9	0.001316	(christmas_supplies)
10	0.001861	(computers)
11	0.068777	(computers_accessories)
12	0.010920	(consoles_games)
13	0.007691	(construction_tools_construction)
14	0.002509	(construction_tools_lights)
15	0.001717	(construction_tools_safety)
16	0.037345	(cool_stuff)
17	0.001995	(costruction_tools_garden)
18	0.003054	(drinks)
19	0.026219	(electronics)
20	0.019166	(fashion_bags_accessories)
21	0.001152	(fashion_male_clothing)
22	0.002468	(fashion_shoes)
23	0.001244	(fashion_underwear_beach)

	support	itemsets
24	0.002231	(fixed_telephony)
25	0.004627	(food)
26	0.002334	(food_drink)
27	0.066310	(furniture_decor)
28	0.004339	(furniture_living_room)
29	0.036173	(garden_tools)
30	0.090853	(health_beauty)
31	0.007856	(home_appliances)
32	0.002406	(home_appliances_2)
33	0.004082	(home_comfort)
34	0.005038	(home_construction)
35	0.060500	(housewares)
36	0.002416	(industry_commerce_and_business)
37	0.002550	(kitchen_dining_laundry_garden_furniture)
38	0.010632	(luggage_accessories)
39	0.002879	(market_place)
40	0.006457	(musical_instruments)

	support	itemsets
41	0.013089	(office_furniture)
42	0.032512	(perfume)
43	0.017582	(pet_shop)
44	0.001439	(signaling_and_security)
45	0.006478	(small_appliances)
46	0.079378	(sports_leisure)
47	0.023762	(stationery)
48	0.043175	(telephony)
49	0.039956	(toys)
50	0.057827	(watches_gifts)
51	0.003218	(books)
52	0.002673	(books, books_technical)

Insight: Lowering the support threshold increased the number of itemsets returned and even yielded itemsets with more than one item.

In [34]:

```
# Apply apriori algorithm to data with min support threshold of 0.00005.
frequent_itemsets = apriori(onehot, min_support = 0.00005, use_colnames =
True)

# Print frequent itemsets.
frequent_itemsets
```

Out[34]:

support	itemsets

	support	itemsets
0	0.001871	(agro_industry_and_commerce)
1	0.002601	(air_conditioning)
2	0.002077	(art)
3	0.000236	(arts_and_crafts)
4	0.003599	(audio)
...	...	...
108	0.000051	(luggage_accessories, stationery)
109	0.000051	(sports_leisure, watches_gifts)
110	0.000144	(sports_leisure, sports_leisure_health_beauty)
111	0.000062	(stationery, toys)
112	0.000144	(sports_leisure, health_beauty, sports_leisure...

113 rows × 2 columns

Observation: Notice how low we must set the support threshold (0.005%) to return a high number of itemsets with more than one item.

In [35]:

```
# Apply apriori algorithm to data with a two-item limit.
frequent_itemsets = apriori(onehot, min_support = 0.00005, max_len = 2,
use_colnames = True)
```

Insight: What do we gain from the apriori algorithm? We start off with 271 potential itemsets and immediately reduce it to 113 without enumerating all 271 itemsets.

Computing association rules from Apriori output

- Use association_rules() to compute and prune association rules from output of apriori().

In [36]:

```
from mlxtend.frequent_patterns import association_rules
```

```
# Recover association rules using support and a minimum threshold of
0.0001.
rules = association_rules(frequent_itemsets, metric = 'support',
min_threshold = 0.0001)

# Print rules header.
rules.head()
```

Out[36]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(baby)	(bed_bath_table)	0.029664	0.096827	0.000175	0.005893	0.060856	-0.002697	0.908527
1	(bed_bath_table)	(baby)	0.096827	0.029664	0.000175	0.001805	0.060856	-0.002697	0.972091
2	(cool_stuff)	(baby)	0.037345	0.029664	0.000206	0.005507	0.185633	-0.000902	0.975709
3	(baby)	(cool_stuff)	0.029664	0.037345	0.000206	0.006932	0.185633	-0.000902	0.969375
4	(baby)	(furniture_decor)	0.029664	0.066310	0.000123	0.004159	0.062728	-0.001844	0.937590

Notice that association_rules automatically computes seven metrics.

Pruning association rules

In [37]:

```
# Recover association rules using confidence threshold of 0.01.
rules = association_rules(frequent_itemsets, metric = 'confidence',
min_threshold = 0.01)

# Print rules.
rules
```

Out[37]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(art)	(furniture_decor)	0.002077	0.066310	0.000051	0.024752	0.373287	-0.000086	0.957388
1	(audio)	(watches_gifts)	0.003599	0.057827	0.000062	0.017143	0.296452	-0.000146	0.958606
2	(furniture_decor)	(bed_bath_table)	0.066310	0.096827	0.000720	0.010854	0.112101	-0.005701	0.913084
3	(home_comfort)	(bed_bath_table)	0.004082	0.096827	0.000442	0.108312	1.118618	0.000047	1.012881
4	(books)	(books_imported)	0.003218	0.000545	0.000545	0.169329	310.722045	0.000543	1.203190
5	(books_imported)	(books)	0.000545	0.003218	0.000545	1.000000	310.722045	0.000543	inf
6	(books)	(books_technical)	0.003218	0.002673	0.002673	0.830671	310.722045	0.002665	5.889872
7	(books_technical)	(books)	0.002673	0.003218	0.002673	1.000000	310.722045	0.002665	inf
8	(construction_tools_lights)	(furniture_decor)	0.002509	0.066310	0.000113	0.045082	0.679872	-0.000053	0.977770
9	(furniture_living_room)	(furniture_decor)	0.004339	0.066310	0.000072	0.016588	0.250155	-0.000216	0.949439

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
10	(home_comfort)	(furniture_decor)	0.004082	0.066310	0.000062	0.015113	0.227921	-0.000209	0.948018
11	(home_construction)	(furniture_decor)	0.005038	0.066310	0.000134	0.026531	0.400103	-0.000200	0.959137
12	(home_construction)	(garden_tools)	0.005038	0.036173	0.000072	0.014286	0.394932	-0.000110	0.977796
13	(sports_leisure_health_beauty)	(health_beauty)	0.000144	0.090853	0.000144	1.000000	11.006790	0.000131	inf
14	(sports_leisure_health_beauty)	(sports_leisure)	0.000144	0.079378	0.000144	1.000000	12.597927	0.000133	inf

In [38]:

```
# Select rules with a consequent support above 0.095.
rules = rules[rules['consequent support'] > 0.095]

# Print rules.
rules
```

Out[38]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(furniture_decor)	(bed_bath_table)	0.066310	0.096827	0.000720	0.010854	0.112101	-0.005701	0.913084
3	(home_comfort)	(bed_bath_table)	0.004082	0.096827	0.000442	0.108312	1.118618	0.000047	1.012881

The leverage metric

- **Leverage provides a sanity check.**
    - �������(����������,����������) = joint support in data.
    - �������(����������)*�������(����������) = expected joint support for unrelated antecedent and consequent.

- **Leverage formula**
    - $$leverage(antecendent, consequent) =

support(antecedent, consequent) - support(antecedent) * support(consequent)$$

- **For most problems, we will discard itemsets with negative leverage.**
    - Negative leverage means that the items appear together less frequently than we would expect if they were randomly and independently distributed across transactions.

In [39]:

```
# Select rules with leverage higher than 0.0.
rules = rules[rules['leverage'] > 0.0]

# Print rules.
rules
```

Out[39]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
3	(home_comfort)	(bed_bath_table)	0.004082	0.096827	0.000442	0.108312	1.118618	0.000047	1.012881

Insight: The Apriori algorithm reduced the number of itemsets from 271 to 113. Pruning allowed us to identify to a single association rule that could be useful for cross-promotional purposes: {h���_�������}→{���_���h_�����}.

Visualizing patterns in metrics

- sns.scatterplot() creates a scatterplot from two columns in a DataFrame.

In [40]:

```
# Recover association rules with a minimum support greater than 0.000001.
rules = association_rules(frequent_itemsets, metric = 'support', 
min_threshold = 0.000001)

# Plot leverage against confidence.
plt.figure(figsize=(15,5))
sns.scatterplot(x="leverage", y="confidence", data=rules)
```

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff4bc0cf828>
```

THANK

YOU