MARKET BASKET ANALYSIS

- INPUT: list of purchases by purchaser
  - do not have names
- ***identify purchase patterns***
  - what items tend to be purchased together
    - obvious: steak-potatoes; beer-pretzels
  - what items are purchased sequentially
    - obvious: house-furniture; car-tires

Importing the libraries

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

This Python code snippet involves importing necessary libraries for data manipulation and visualization. Here's the explanation:

1. `import numpy as np`: This line imports the NumPy library and assigns it the alias `np`. NumPy is a powerful library for numerical computing in Python. By convention, it's imported as `np` to save typing and make the code more readable.
2. `import matplotlib.pyplot as plt`: This line imports the `pyplot` module from the Matplotlib library and assigns it the alias `plt`. Matplotlib is a widely used plotting library in Python. The `pyplot` module provides a MATLAB-like plotting interface. Importing it as `plt` is a common convention.
3. `import pandas as pd`: This line imports the Pandas library and assigns it the alias `pd`. Pandas is a powerful library for data manipulation and analysis. Importing it as `pd` is a common convention.

By importing these libraries, you gain access to a wide range of functions and tools for numerical computing, data manipulation, and visualization, which are essential for many data science and machine learning tasks.

Loading Data

```
#importing libraries
dataset = pd.read_csv('/content/Market_Basket_Optimisation.csv.xls', lo
#viewing dataset
dataset
```

	0	1	2	3	4	5	6	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottag chees
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	Na
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	Na
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	Na
...	...	...	...	...	...	...	...	
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	Na
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	Na
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	Na
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	Na
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	Na

7501 rows × 20 columns

This code snippet deals with loading and viewing a dataset in Python. Here's a breakdown:

1. **Importing pandas library (`pd`)**: This line assumes `pandas` is already installed (if not, use `pip install pandas` in your terminal). Pandas is a powerful library for data manipulation and analysis in Python.
2. **Loading the dataset (`dataset = pd.read_csv(...)`):**
    - `pd.read_csv` is a function from pandas used to read data from CSV (comma-separated values) files. Here, it's likely reading a file named 'Market_Basket_Optimisation.csv.xls' (which is a bit unusual as the extension is `.xls` for Excel files and `.csv` for CSV files).
    - `'/content/Market_Basket_Optimisation.csv.xls'` specifies the file path. Make sure this file is in the same directory (folder) as your Python code or adjust the path accordingly.
    - `low_memory=False` (optional argument here) is set to avoid potential memory issues when reading large datasets.
    - `header=None` (another optional argument) suggests the CSV file might not have a header row (the first row containing column names). If your file does have a header row, remove this argument.
3. **Viewing the dataset (`dataset`)**: The final line simply prints the content of the variable `dataset`. Since `dataset` now holds the loaded data (most likely a Pandas DataFrame), this will display a limited preview of the data, usually the first few rows and columns.

To see more of the data, you can use methods like:

- `.head(n)`: Print the first `n` rows (default is 5).
- `.tail(n)`: Print the last `n` rows (default is 5).

I hope this concise explanation is helpful!

Understanding Data

```
dataset.head()
```

	0	1	2	3	4	5	6	7	8
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN

The `dataset.head()` function call is used to display the first few rows of the DataFrame `dataset`. Here's an explanation:

1. `dataset`: This is the Pandas DataFrame containing the dataset that was previously read from the CSV file.
2. `.head()`: This is a method in Pandas used to retrieve the first few rows of a DataFrame. By default, it returns the first 5 rows, but you can specify the number of rows you want to retrieve by passing an argument to the function. For example, `dataset.head(10)` would return the first 10 rows of the dataset.

By calling `dataset.head()`, you're essentially displaying a preview of the dataset, which helps in understanding its structure and the type of data it contains. This is often used for exploratory data analysis to get a quick overview of the data before performing further operations or analysis.

```
dataset.shape
```

```
(7501, 20)
```

The `dataset.shape` attribute is used to retrieve the dimensions of the DataFrame `dataset`. Here's the explanation:

1. `dataset`: This is the Pandas DataFrame containing the dataset that was previously read from the CSV file or generated in the code.
2. `.shape`: This is an attribute in Pandas DataFrames which returns a tuple representing the dimensions of the DataFrame. The tuple returned has two elements: the number of rows and the number of columns, in that order.

So, when you call `dataset.shape`, you're retrieving the number of rows and columns in the DataFrame. This is useful for understanding the size of the dataset and its structure, especially when dealing with larger datasets or when you need to perform operations that depend on the dimensions of the DataFrame.

```
#basic information of dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7501 entries, 0 to 7500
Data columns (total 20 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       7501 non-null   object
 1   1       5747 non-null   object
 2   2       4389 non-null   object
 3   3       3345 non-null   object
 4   4       2529 non-null   object
 5   5       1864 non-null   object
 6   6       1369 non-null   object
 7   7       981 non-null    object
 8   8       654 non-null    object
 9   9       395 non-null    object
 10  10      256 non-null    object
 11  11      154 non-null    object
 12  12      87 non-null     object
 13  13      47 non-null     object
 14  14      25 non-null     object
 15  15      8 non-null      object
 16  16      4 non-null      object
 17  17      4 non-null      object
 18  18      3 non-null      object
 19  19      1 non-null      object
dtypes: object(20)
memory usage: 1.1+ MB
```

The `dataset.info()` method provides basic information about the DataFrame `dataset`, including the data types of each column and the number of non-null values. Here's an explanation:

1. `dataset`: This is the Pandas DataFrame containing the dataset that was previously read from the CSV file or generated in the code.
2. `.info()`: This method in Pandas DataFrame provides a concise summary of the DataFrame's structure and content. It includes the following information:
    - The total number of entries (rows) in the DataFrame.
    - The data type of each column.
    - The number of non-null values in each column.
    - Additional memory usage information.

By calling `dataset.info()`, you get a summary of the dataset, which helps you understand the data types of the columns and identify any missing values. This information is crucial for data preprocessing and analysis, as it gives you insights into the nature and quality of the dataset.

```
#basic info of statistics
dataset.describe()
```

	0	1	2	3	4	5	6	7	8	
count	7501	5747	4389	3345	2529	1864	1369	981	654	
unique	115	117	115	114	110	106	102	98	88	
top	mineral water	mineral water	mineral water	mineral water	green tea	french fries	green tea	green tea	green tea	gı
freq	577	484	375	201	153	107	96	67	57	

The `dataset.describe()` method provides basic statistics for numerical columns in the DataFrame `dataset`. Here's an explanation:

1. `dataset`: This is the Pandas DataFrame containing the dataset that was previously read from the CSV file or generated in the code.
2. `.describe()`: This method in Pandas DataFrame computes descriptive statistics for all numerical columns in the DataFrame. It includes statistics such as count, mean, standard deviation, minimum, maximum, and various quantiles (25th, 50th, and 75th percentiles).

By calling `dataset.describe()`, you get a summary of basic statistics for numerical columns in the dataset. This summary provides insights into the central tendency, dispersion, and distribution of the numerical data. It's useful for understanding the range and distribution of values in each numerical column, identifying outliers, and gaining initial insights into the dataset's numerical characteristics.

```
#viewing the uniqueness in dataset
dataset.nunique()
```

```
0     115
1     117
2     115
3     114
4     110
5     106
6     102
7      98
8      88
9      80
10     66
11     50
12     43
13     28
14     19
15      8
16      3
17      3
18      3
19      1
dtype: int64
```

The `dataset.nunique()` method is used to count the number of unique values in each column of the DataFrame `dataset`. Here's an explanation:

1. `dataset`: This is the Pandas DataFrame containing the dataset that was previously read from the CSV file or generated in the code.
2. `.nunique()`: This method in Pandas DataFrame computes the number of unique values for each column. It returns a Series containing the count of unique values for each column, with the column names as the index.

By calling `dataset.nunique()`, you get the count of unique values for each column in the dataset. This information is useful for understanding the diversity of values within each column, identifying categorical variables, and assessing the uniqueness of data in the dataset. It's particularly helpful in data preprocessing and analysis tasks where understanding the distribution of unique values is essential.

Missing Values

```
#missing values any from the dataset
print(str('Any missing data or NaN in the dataset:'), dataset.isnull().
```

```
Any missing data or NaN in the dataset: True
```

This code snippet checks whether there are any missing values (NaN) in the dataset. Here's an explanation:

1. `print()`: This is a Python built-in function used to display output.
2. `str('Any missing data or NaN in the dataset:')`: This part creates a string that serves as a message to be displayed.
3. `dataset.isnull().values.any()`:
    - `dataset.isnull()`: This returns a DataFrame of the same shape as `dataset`, where each element is True if the corresponding element in `dataset` is NaN, and False otherwise.
    - `.values`: This converts the DataFrame of boolean values into a NumPy array.
    - `.any()`: This method checks if any of the values in the array are True. If any value is True, it returns True; otherwise, it returns False.

The entire expression checks if there are any missing values (NaN) in the dataset. If any missing value is found, it will print `True`, indicating that there are missing values. Otherwise, it will print `False`, indicating that there are no missing values.

So, the printed output will indicate whether there are any missing values (NaN) in the dataset.

Apirori Algorithm

```
#install APRIORI algorithm
!pip install apyori
```

```
Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl s
  Stored in directory: /root/.cache/pip/wheels/c4/1a/79/20f55c470a50
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

The code snippet you provided utilizes a command that's typically used in Jupyter Notebook or Google Colab environments to install Python packages via pip. Here's an explanation:

1. `!pip install apyori`: This is a command-line instruction preceded by an exclamation mark (`!`). In Jupyter Notebook or Google Colab, this notation is used to execute shell commands from within the notebook environment.
2. `pip install apyori`: This command installs the Python package named `apyori`. `apyori` is a library that implements the Apriori algorithm, which is a popular algorithm for mining frequent itemsets and association rules in a dataset.

By running `!pip install apyori`, you're instructing the notebook environment to install the `apyori` package from the Python Package Index (PyPI). Once installed, you'll be able to use the Apriori algorithm functionality provided by the `apyori` library in your code.

```
#Let's create an empty list here
list_of_transactions = []
```

This code snippet creates an empty list named `list_of_transactions`. Here's an explanation:

1. `list_of_transactions`: This is the name given to the empty list that is being created.
2. `=[]`: This syntax initializes an empty list. By using square brackets with no elements inside (`[]`), an empty list is created.

By executing this code, you're creating a variable (`list_of_transactions`) that can be used to store transaction data. Typically, in association rule mining tasks such as using the Apriori algorithm, transaction data is stored in the form of lists, where each list represents a transaction and contains the items bought or actions taken within that transaction. This empty list can later be populated with transaction data as needed.

```
#Append the list
for i in range(0, 7501):
    list_of_transactions.append([str(dataset.values[i,j]) for j in rang
```

This code snippet iterates over the rows of the dataset and appends each row (transaction) as a list to the `list_of_transactions`. Here's an explanation:

1. `for i in range(0, 7501):`: This loop iterates over the range from 0 to 7500 (exclusive). It indicates that the loop will execute 7501 times, covering each row of the dataset.
2. `list_of_transactions.append([...])`: Within the loop, each row of the dataset is converted into a list (transaction) and appended to the `list_of_transactions`.
3. `[str(dataset.values[i,j]) for j in range(0, 20)]`: This is a list comprehension that generates a list containing the values of each cell in the current row (`i`) of the dataset. It iterates over the columns of the dataset (from 0 to 19) and converts each value to a string using `str()`. This list represents a single transaction.
    - `dataset.values[i, j]` accesses the value at the i-th row and j-th column of the dataset.
    - `str(dataset.values[i, j])` converts the value to a string.
    - The list comprehension `[... for j in range(0, 20)]` iterates over the columns and creates a list of values for the current row.

Overall, this code constructs a list of transactions (`list_of_transactions`), where each transaction is represented as a list of items from the corresponding row of the dataset. This format is suitable for further processing, such as applying association rule mining algorithms like Apriori.

```
#Let's see the first element from our list of transactions. We should i
list_of_transactions[0]
```

```
['shrimp',
 'almonds',
 'avocado',
 'vegetables mix',
 'green grapes',
 'whole weat flour',
 'yams',
 'cottage cheese',
 'energy drink',
 'tomato juice',
 'low fat yogurt',
 'green tea',
 'honey',
 'salad',
 'mineral water',
 'salmon',
 'antioxydant juice',
 'frozen smoothie',
 'spinach',
 'olive oil']
```

This code snippet accesses the first element (transaction) from the `list_of_transactions` list. Here's an explanation:

1. `list_of_transactions[0]`: This expression retrieves the first element (at index 0) from the `list_of_transactions` list. In Python, indexing starts from 0, so `list_of_transactions[0]` refers to the first element in the list.

By executing this code, you'll see the first transaction stored in the `list_of_transactions`. This transaction is represented as a list containing the items or actions from the first row of the dataset.

Training Apirori Algorithm

```
# Training apiori algorithm on our list_of_transactions
from apyori import apriori
rules = apriori(list_of_transactions, min_support = 0.004, min_confiden
```

This code snippet demonstrates training the Apriori algorithm on the `list_of_transactions` using the `apyori` library. Here's an explanation:

1. `from apyori import apriori`: This line imports the `apriori` function from the `apyori` module. The `apyori` library is used to implement the Apriori algorithm for association rule mining.
2. `rules = apriori(list_of_transactions, min_support=0.004, min_confidence=0.2, min_lift=3, min_length=2)`: This line trains the Apriori algorithm on the list of transactions (`list_of_transactions`) using the `apriori` function. The function takes several parameters:
    - `list_of_transactions`: This is the list of transactions on which the Apriori algorithm will be trained.
    - `min_support`: This parameter specifies the minimum support threshold for an itemset to be considered frequent. Support is the proportion of transactions in the dataset that contain the itemset.
    - `min_confidence`: This parameter specifies the minimum confidence threshold for association rules to be generated. Confidence is the probability that the rule {A} -> {B} holds given {A}.
    - `min_lift`: This parameter specifies the minimum lift threshold for association rules to be generated. Lift measures the ratio of the observed support to that expected if the two items were independent. It indicates the strength of the association between the antecedent and the consequent.
    - `min_length`: This parameter specifies the minimum number of items in an itemset or an association rule.

    The `apriori` function returns a generator object containing the association rules mined from the dataset based on the specified parameters. These rules can be further analyzed or used for making recommendations or insights.

By executing this code, you'll generate association rules based on the specified parameters from the transactions in `list_of_transactions`.

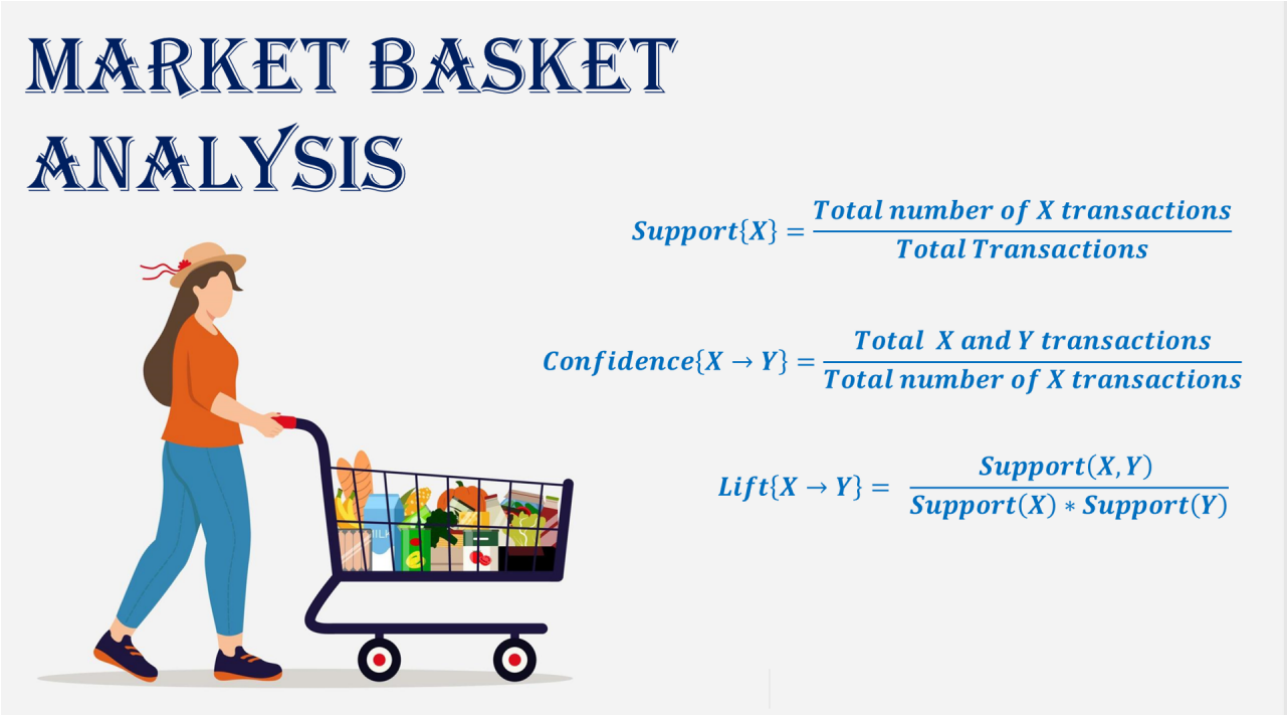
```
# Create a list of rules and print the results
results = list(rules)
#Here is the first rule in list or results
results[0]
```

```
RelationRecord(items=frozenset({'light cream', 'chicken'}),
support=0.004532728969470737, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'light cream'}),
items_add=frozenset({'chicken'}), confidence=0.29059829059829057,
lift=4.84395061728395)])
```

This code snippet creates a list of association rules (`results`) generated by the Apriori algorithm and prints the first rule in the list. Here's an explanation:

1. `results = list(rules)` : This line converts the generator object `rules` into a list named `results` . The generator object contains association rules mined by the Apriori algorithm.
2. `results[0]` : This line accesses the first rule in the `results` list. In Python, indexing starts from 0, so `results[0]` refers to the first rule in the list.

By executing this code, you'll obtain the first association rule generated by the Apriori algorithm from the list of rules (`results`). This rule will typically be in the format of antecedent -> consequent with associated support, confidence, and lift values, providing insights into the relationships between items in the dataset.



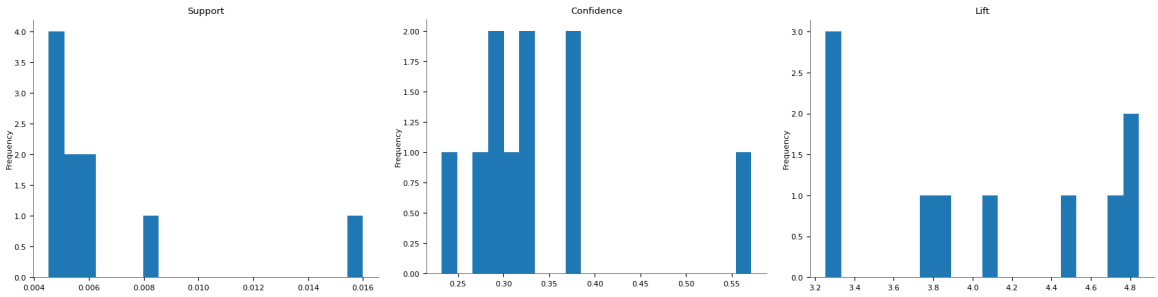
Bought Together Items

```
#In order to visualize our rules better we need to extract elements fro
#Here is the code for this. We have extracted left hand side and right

def inspect(results):
    lhs     =  [tuple(result [2] [0] [0]) [0] for result in results]
    rhs     =  [tuple(result [2] [0] [1]) [0] for result in results]
    supports = [result [1] for result in results]
    confidences = [result [2] [0] [2]   for result in results]
    lifts = [result [2] [0] [3]   for result in results]
    return list(zip(lhs,rhs,supports,confidences, lifts))
resultsinDataFrame = pd.DataFrame(inspect(results),columns = ['Left Han
resultsinDataFrame.head(10)
```

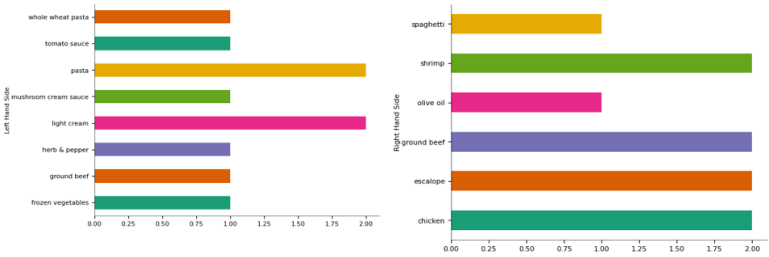
	Left Hand Side	Right Hand Side	Support	Confidence	Lift
0	light cream	chicken	0.004533	0.290598	4.843951
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
2	pasta	escalope	0.005866	0.372881	4.700812
3	herb & pepper	ground beef	0.015998	0.323450	3.291994

4	tomato sauce	ground beef	0.005333	0.377358	3.840659
5	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
6	pasta	shrimp	0.005066	0.322034	4.506672
7	light cream	chicken	0.004533	0.290598	4.843951
8	frozen vegetables	shrimp	0.005333	0.232558	3.254512
9	ground beef	spaghetti	0.004799	0.571429	3.281995

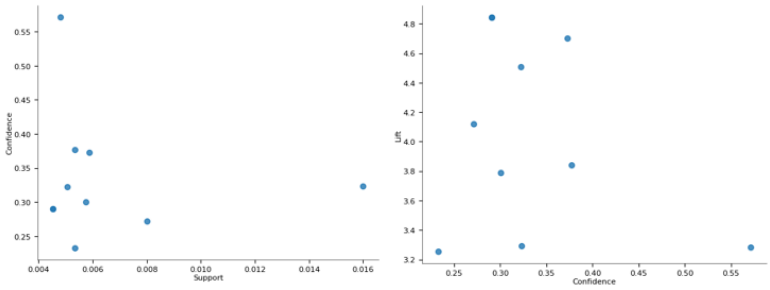
Distributions



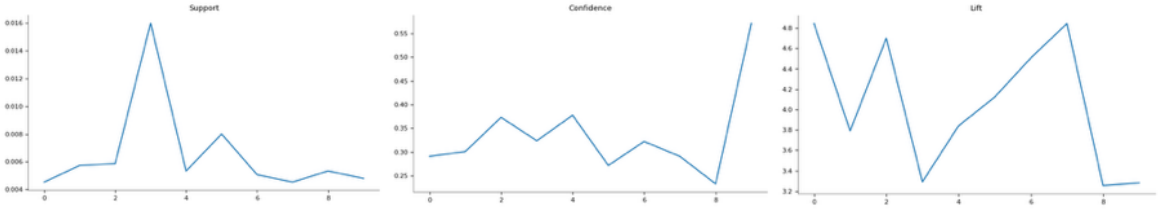
Categorical distributions



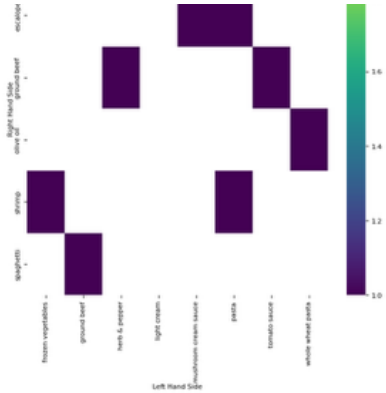
2-d distributions



Values

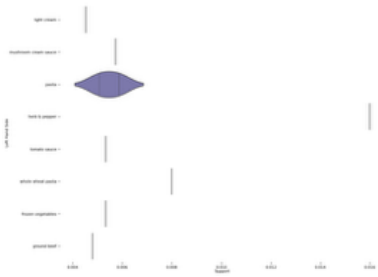


2-d categorical distributions

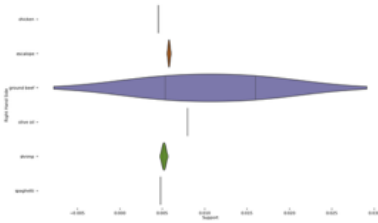


Faceted distributions

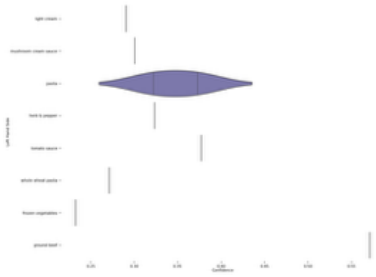
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
```



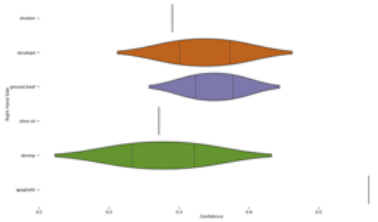
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is d
```



```
<string>:5: FutureWarnin
Passing `palette` without
```



```
<st
Pas
```



This code defines a function `inspect(results)` to extract elements from the association rules generated by the Apriori algorithm, convert them into a pandas DataFrame, and sort the rules by the lift value. Here's an explanation:

1. `def inspect(results)`: This line defines a function named `inspect` that takes the `results` list of association rules as input.
2. Inside the `inspect` function:
    - `lhs`: This line extracts the left-hand side (antecedent) items from each association rule.
    - `rhs`: This line extracts the right-hand side (consequent) items from each association rule.
    - `supports`: This line extracts the support values from each association rule.
    - `confidences`: This line extracts the confidence values from each association rule.
    - `lifts`: This line extracts the lift values from each association rule.
    - Finally, the function returns a list of tuples containing the extracted information for each association rule.
3. `resultsinDataFrame = pd.DataFrame(inspect(results),columns = ['Left Hand Side', 'Right Hand Side', 'Support', 'Confidence', 'Lift'])`: This line converts the list of tuples returned by the `inspect` function into a pandas DataFrame named `resultsinDataFrame`. The columns of the DataFrame are labeled as 'Left Hand Side', 'Right Hand Side', 'Support', 'Confidence', and 'Lift'.
4. `resultsinDataFrame.head(10)`: This line prints the first 10 rows of the DataFrame `resultsinDataFrame`, allowing you to visualize the extracted association rules in a tabular format.

By executing this code, you'll get a DataFrame containing the extracted information from the association rules, making it easier to analyze and interpret the results. Each row represents an association rule, with columns indicating the left-hand side item, right-hand side item, support, confidence, and lift values.

Frequent Items

```
#As we have our rules in pd.dataframe we can sort it by lift value usin
resultsinDataFrame.nlargest(n=10, columns='Lift')
```

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
0	light cream	chicken	0.004533	0.290598	4.843951
7	light cream	chicken	0.004533	0.290598	4.843951
2	pasta	escalope	0.005866	0.372881	4.700812
12	pasta	escalope	0.005866	0.372881	4.700812
30	pasta	shrimp	0.005066	0.322034	4.515096
6	pasta	shrimp	0.005066	0.322034	4.506672
36	frozen vegetables	spaghetti	0.004399	0.259843	4.350622
55	frozen vegetables	spaghetti	0.004399	0.259843	4.350622
10	ground beef	herb & pepper	0.004133	0.206667	4.178455
35	ground beef	herb & pepper	0.004133	0.206667	4.178455

The `nlargest()` method in pandas DataFrame allows you to retrieve the top N rows with the largest values in a specified column.

```
resultsinDataFrame.nlargest(n=10, columns='Lift')
```

This line of code retrieves the top 10 rows from the `resultsinDataFrame` DataFrame based on the values in the 'Lift' column. Here's an explanation:

- `n=10`: Specifies that you want to retrieve the top 10 rows.
- `columns='Lift'`: Specifies that you want to consider the 'Lift' column for sorting.

By executing this code, you'll get the top 10 association rules sorted by the 'Lift' value, allowing you to focus on the rules with the strongest association based on lift. Each row represents an association rule, with columns indicating the left-hand side item, right-hand side item, support, confidence, and lift values.