Experiment 8 - Recommendation

| Roll No. |  |
| --- | --- |
| Name |  |
| Class | D15A |
| Subject | DS using Python Lab |
| LO Mapped | LO2:  Understand the concept of Data science process and associated terminologies to solve real-world problems .  LO4: Apply the different unsupervised machine learning algorithms like Clustering or Association to solve the problems. |
|  |  |

**Aim**:

To implement the Association Mining algorithm using Python.

**Association Mining Algorithm**:

**Association Mining:**

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of the dataset. It is based on different rules to discover the interesting relations between variables in the database.

The association rule learning is one of the very important concepts of machine learning, and it is employed in Market Basket analysis, Web usage mining, continuous production, etc. Here market basket analysis is a technique used by the various big retailers to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

Market Based Analysis is one of the key techniques used by large relations to show associations between items.It allows retailers to identify relationships between the items that people buy together frequently.

Given a set of transactions, we can find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.

**TID Items**

1 Bread, Milk

2 Bread, Diaper, Beer, Eggs

3 Milk, Diaper, Beer, Coke

4 Bread, Milk, Diaper, Beer

5 Bread, Milk, Diaper, Coke

Before we start defining the rule, let us first see the basic definitions.

**Support Count** – Frequency of occurrence of a itemset.

Here Support count({Milk, Bread, Diaper})=2

**Frequent Itemset** – An itemset whose support is greater than or equal to minsup threshold.

**Association Rule** – An implication expression of the form X -> Y, where X and Y are any 2 itemsets.

Example: {Milk, Diaper}->{Beer}

**Rule Evaluation Metrics –**

Support(s) –

The number of transactions that include items in the {X} and {Y} parts of the rule as a percentage of the total number of transaction.It is a measure of how frequently the collection of items occur together as a percentage of all transactions.

Support = support count(X+Y) / total –

It is interpreted as a fraction of transactions that contain both X and Y.

Confidence(c) –

It is the ratio of the no of transactions that includes all items in {B} as well as the no of transactions that includes all items in {A} to the no of transactions that includes all items in {A}.

Conf(X=>Y) = Supp(XUY) / Supp(X) –

It measures how often each item in Y appears in transactions that contain items in X also.

Lift(l) –

The lift of the rule X=>Y is the confidence of the rule divided by the expected confidence, assuming that the itemsets X and Y are independent of each other.The expected confidence is the confidence divided by the frequency of {Y}.

Lift(X=>Y) = Conf(X=>Y) \div Supp(Y) –

Lift value near 1 indicates X and Y almost often appear together as expected, greater than 1 means they appear together more than expected and less than 1 means they appear less than expected.Greater lift values indicate stronger association.

**Types of Association Rule Learning**

Association rule learning can be divided into three algorithms:

1. Apriori Algorithm

This algorithm uses frequent datasets to generate association rules. It is designed to work on the databases that contain transactions. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset efficiently.

It is mainly used for market basket analysis and helps to understand the products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

2. Eclat Algorithm

The Eclat algorithm stands for Equivalence Class Transformation. This algorithm uses a depth-first search technique to find frequent itemsets in a transaction database. It performs faster than the Apriori Algorithm.

3. F-P Growth Algorithm

The F-P growth algorithm stands for Frequent Pattern, and it is the improved version of the Apriori Algorithm. It represents the database in the form of a tree structure that is known as a frequent pattern or tree. The purpose of this frequent tree is to extract the most frequent patterns.

**Applications of Association Rule Learning**

It has various applications in machine learning and data mining. Below are some popular applications of association rule learning:

1. Market Basket Analysis: It is one of the popular examples and applications of association rule mining. This technique is commonly used by big retailers to determine the association between items.
2. Medical Diagnosis: With the help of association rules, patients can be cured easily, as it helps in identifying the probability of illness for a particular disease.
3. Protein Sequence: The association rules help in determining the synthesis of artificial Proteins.
4. It is also used for Catalog Design and Loss-leader Analysis and many more other applications.

**Python Library Function Used**:

We have used **apriori** and **association\_rules** functions of the **mlxtend.frequent\_patterns** package.

**1. apriori**

*apriori(df, min\_support=0.5, use\_colnames=False, max\_len=None, verbose=0, low\_memory=False)*

Get frequent itemsets from a one-hot DataFrame

**Parameters**

* df : pandas DataFrame  
  pandas DataFrame the encoded format. Also supports DataFrames with sparse data; for more info, please see (https://pandas.pydata.org/pandas-docs/stable/ user\_guide/sparse.html#sparse-data-structures)  
  Please note that the old pandas SparseDataFrame format is no longer supported in mlxtend >= 0.17.2.  
  The allowed values are either 0/1 or True/False. For example,  
  Apple Bananas Beer Chicken Milk Rice 0 True False True True False True 1 True False True False False True 2 True False True False False False 3 True True False False False False 4 False False True True True True 5 False False True False True True 6 False False True False True False 7 True True False False False False
* min\_support : float (default: 0.5)  
  A float between 0 and 1 for minumum support of the itemsets returned. The support is computed as the fraction transactions\_where\_item(s)\_occur / total\_transactions.
* use\_colnames : bool (default: False)  
  If True, uses the DataFrames' column names in the returned DataFrame instead of column indices.
* max\_len : int (default: None)  
  Maximum length of the itemsets generated. If None (default) all possible itemsets lengths (under the apriori condition) are evaluated.
* verbose : int (default: 0)  
  Shows the number of iterations if >= 1 and low\_memory is True. If  
  =1 and low\_memory is False, shows the number of combinations.
* low\_memory : bool (default: False)  
  If True, uses an iterator to search for combinations above min\_support. Note that while low\_memory=True should only be used for large dataset if memory resources are limited, because this implementation is approx. 3-6x slower than the default.

**Returns**

pandas DataFrame with columns ['support', 'itemsets'] of all itemsets that are >= min\_support and < than max\_len (if max\_len is not None). Each itemset in the 'itemsets' column is of type frozenset, which is a Python built-in type that behaves similarly to sets except that it is immutable (For more info, see https://docs.python.org/3.6/library/stdtypes.html#frozenset).

2. **association\_rules**

*association\_rules(df, metric='confidence', min\_threshold=0.8, support\_only=False)*

Generates a DataFrame of association rules including the metrics 'score', 'confidence', and 'lift'

**Parameters**

* df : pandas DataFrame  
  pandas DataFrame of frequent itemsets with columns ['support', 'itemsets']
* metric : string (default: 'confidence')  
  Metric to evaluate if a rule is of interest. **Automatically set to 'support' if support\_only=True.** Otherwise, supported metrics are 'support', 'confidence', 'lift',

'leverage', and 'conviction'

* min\_threshold : float (default: 0.8)  
  Minimal threshold for the evaluation metric, via the metric parameter, to decide whether a candidate rule is of interest.
* support\_only : bool (default: False)  
  Only computes the rule support and fills the other metric columns with NaNs. This is useful if:  
  a) the input DataFrame is incomplete, e.g., does not contain support values for all rule antecedents and consequents  
  b) you simply want to speed up the computation because you don't need the other metrics.

**Returns**

pandas DataFrame with columns "antecedents" and "consequents" that store itemsets, plus the scoring metric columns: "antecedent support", "consequent support", "support", "confidence", "lift", "leverage", "conviction" of all rules for which metric(rule) >= min\_threshold. Each entry in the "antecedents" and "consequent" columns are of type frozenset, which is a Python built-in type that behaves similarly to sets except that it is immutable.

**Data Modeling and Analysis**

**Dataset**: Movies Dataset

**Preprocessing**:

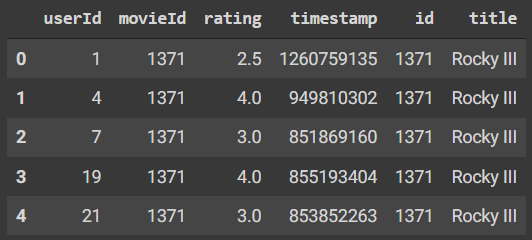
title\_mask = movies\_df['title'].isna()

movies\_df = movies\_df.loc[title\_mask == False]

movies\_df = movies\_df.astype({'id': 'int64'})

df = pd.merge(ratings\_df, movies\_df[['id', 'title']], left\_on='movieId', right\_on='id')

df.head()



Id column is repeated and the timestamp is not important for this problem. So, you can drop the two.

df.drop(['timestamp', 'id'], axis=1, inplace=True)

Dropping duplicates

df = df.drop\_duplicates(['userId','title'])

Feature Scaling:

df\_pivot = df.pivot(index='userId', columns='title', values='rating').fillna(0)

df\_pivot = df\_pivot.astype('int64')

def encode\_ratings(x):

if x<=0:

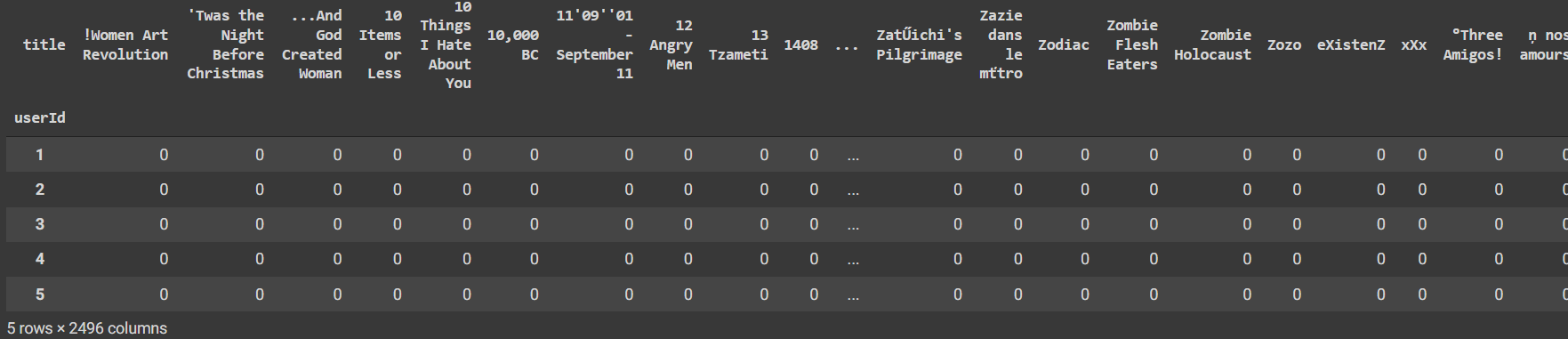
return 0

if x>=1:

return 1

df\_pivot = df\_pivot.applymap(encode\_ratings)

df\_pivot.head()



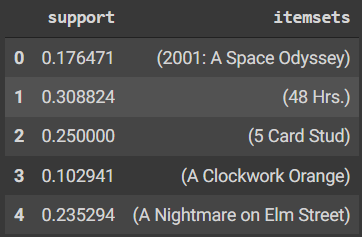
**Code and Observation**:

**Using library functions:**

from mlxtend.frequent\_patterns import apriori

frequent\_itemset = apriori(df\_pivot, min\_support=0.07, use\_colnames=True)

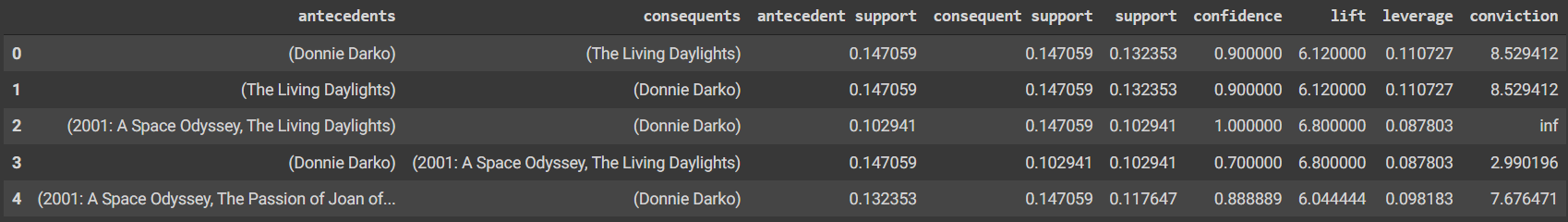
frequent\_itemset.head()



from mlxtend.frequent\_patterns import association\_rules

rules = association\_rules(frequent\_itemset, metric="lift", min\_threshold=1)

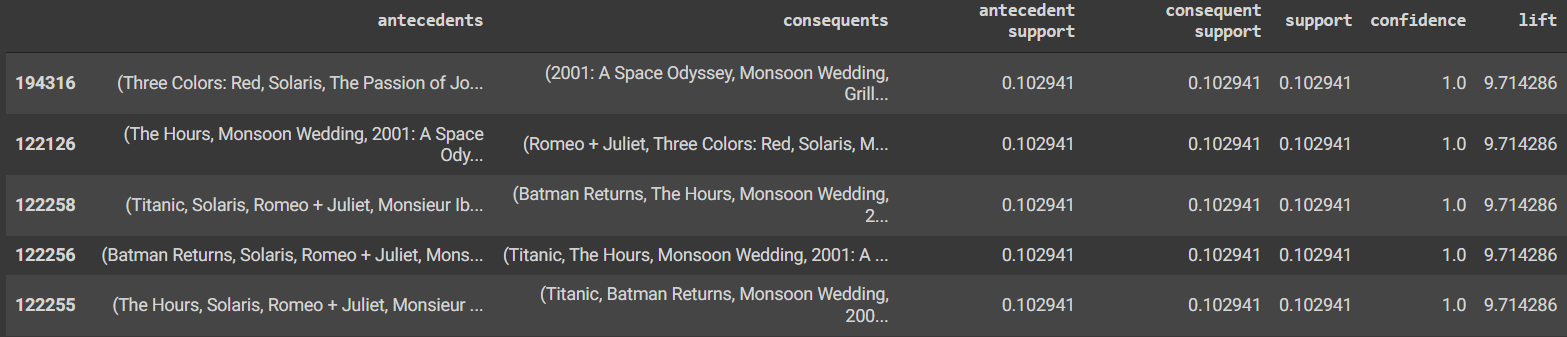
rules.head()



Metrics:

df\_res = rules.sort\_values(by=['lift'], ascending=False)

df\_res.head()



print("Number of rules generated:",len(df\_res))

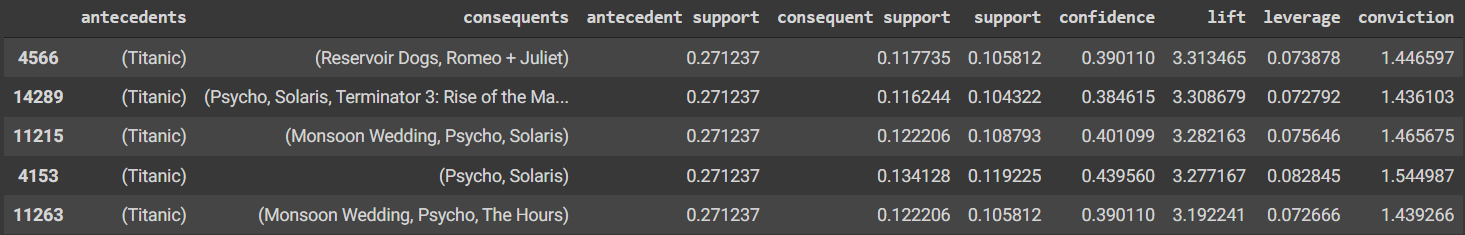


Recommendations for "Titanic":

df\_T = df\_res[df\_res['antecedents'].apply(lambda x: len(x) ==1 and next(iter(x)) == 'Titanic')]

df\_T = df\_T[df\_T['lift'] > 2]

df\_T.head()



movies = df\_T['consequents'].values

movie\_list = []

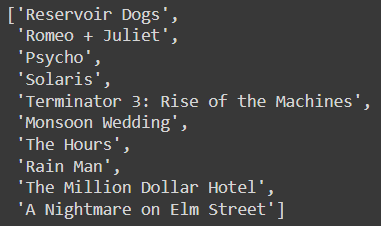
for movie in movies:

for title in movie:

if title not in movie\_list:

movie\_list.append(title)

movie\_list[0:10]



**Using user defined functions:**

def APRIORI\_MY(data, min\_support=0.07, max\_length = 4):

# Collecting Required Library

import numpy as np

import pandas as pd

from itertools import combinations

# Step 1:

# Creating a dictionary to stored support of an itemset.

support = {}

L = list(data.columns)

# Step 2:

#generating combination of items with len i in ith iteration

for i in range(1, max\_length+1):

c = set(combinations(L,i))

# Reset "L" for next ith iteration

L =set()

# Step 3:

#iterate through each item in "c"

print(i,c)

for j in list(c):

# print(j)

sup = data.loc[:,j].product(axis=1).sum()/len(data.index)

if sup > min\_support:

#print(sup, j)

support[j] = sup

# Appending frequent itemset in list "L", already reset list "L"

L = list(set(L) | set(j))

# Step 4: data frame with cols "items", 'support'

result = pd.DataFrame(list(support.items()), columns = ["Items", "Support"])

return(result)

my\_freq\_itemset = APRIORI\_MY(df\_pivot, 0.1, 4)

my\_freq\_itemset.sort\_values(by = 'Support', ascending = False)



def ASSOCIATION\_RULE\_MY(df, min\_threshold=0.5):

import pandas as pd

from itertools import permutations

# STEP 1:

#creating required varaible

support = pd.Series(df.Support.values, index=df.Items).to\_dict()

data = []

L= df.Items.values

# Step 2:

#generating rule using permutation

p = list(permutations(L, 2))

# Iterating through each rule

for i in p:

# If LHS(Antecedent) of rule is subset of RHS then valid rule.

if set(i[0]).issubset(i[1]):

conf = support[i[1]]/support[i[0]]

#print(i, conf)

if conf > min\_threshold:

#print(i, conf)

j = i[1][not i[1].index(i[0][0])]

lift = support[i[1]]/(support[i[0]]\* support[(j,)])

leverage = support[i[1]] - (support[i[0]]\* support[(j,)])

convection = (1 - support[(j,)])/(1- conf)

data.append([i[0], (j,), support[i[0]], support[(j,)], support[i[1]], conf, lift, leverage, convection])

# STEP 3:

result = pd.DataFrame(data, columns = ["antecedents", "consequents", "antecedent support", "consequent support",

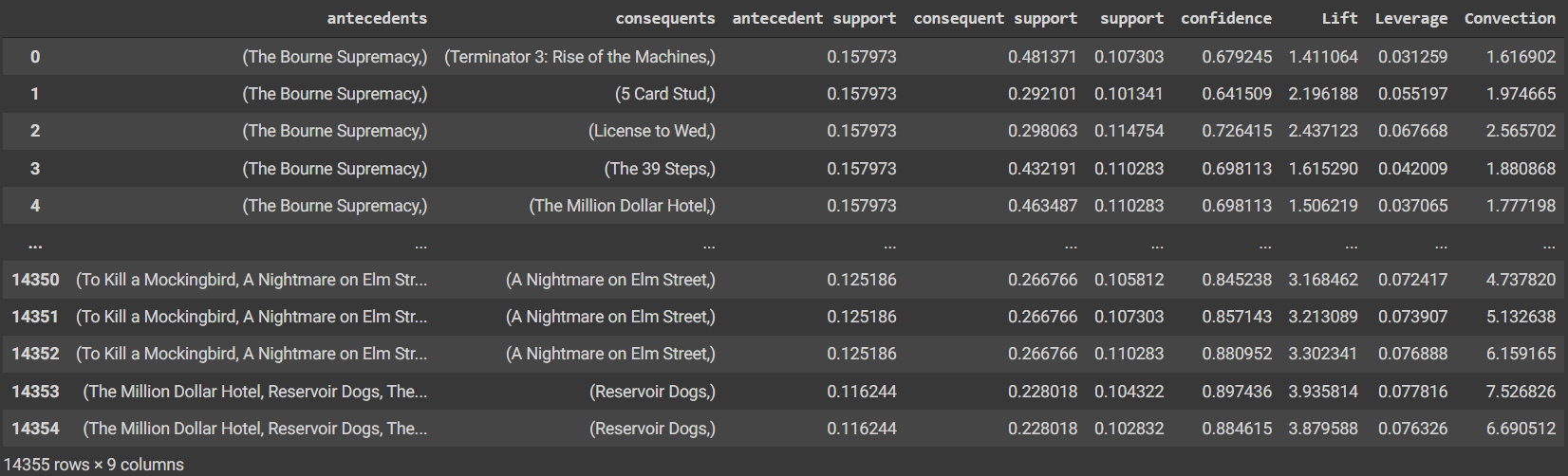
"support", "confidence", "Lift", "Leverage", "Convection"])

return(result)

## Rule with minimun confidence = 50%

my\_rule = ASSOCIATION\_RULE\_MY(my\_freq\_itemset, 0.5)

my\_rule



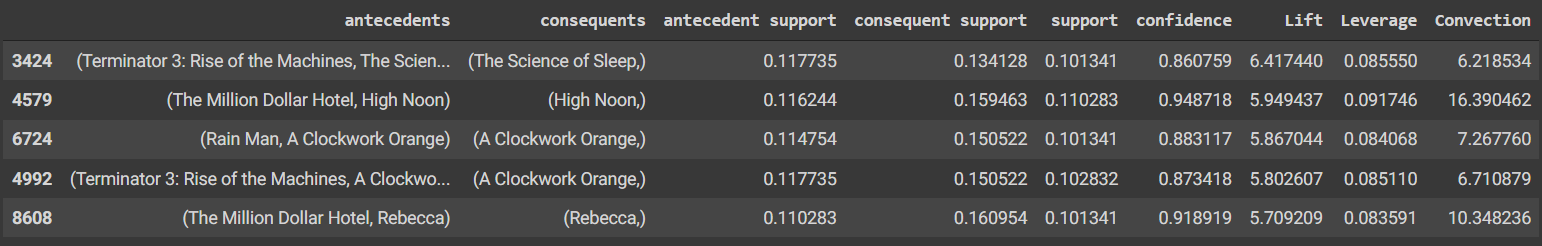
Metrics:

print("Number of rules generated:",len(my\_rule))



df\_my\_rules = my\_rule.sort\_values(by='Lift', ascending= False).head(10)

df\_my\_rules.head()

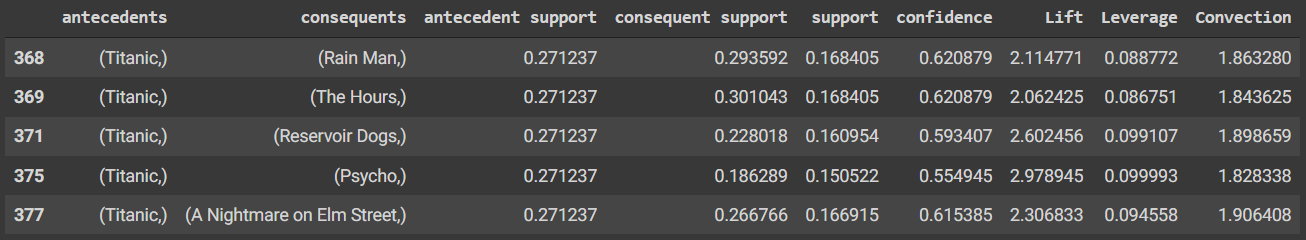


Recommendations of "Titanic":

df\_T = my\_rule[my\_rule['antecedents'].apply(lambda x: len(x) ==1 and next(iter(x)) == 'Titanic')]

df\_T = df\_T[df\_T['Lift'] > 2]

df\_T.head()



movies = df\_T['consequents'].values

movie\_list = []

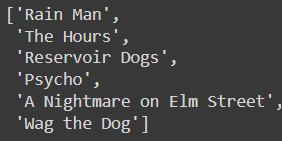
for movie in movies:

for title in movie:

if title not in movie\_list:

movie\_list.append(title)

movie\_list[0:10]



**Conclusion**:

Thus, we have learnt about association mining, its algorithms and also learnt how to implement the algorithms using python libraries and user-defined functions.