Creating A Synthetic Dataset for Association Rule Learning with the Apriori Algorithm- Marine Seismic Example

- Apriori[1] is an algorithm for frequent item set mining and association rule learning over relational databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis [2].

- The Apriori algorithm is generally applied for finding "item combinations are frequently seem together".

- In this study, I want to show you how to create a dataset for association rule learning on Marine geoscience interpretation

Marine Seismic Interpretation Example

Story: I am a marine geoscientist who interprets marine structures and related anomalies on seismic data in the Sea of Marmara. A lot of data came in and I interpreted them and created a database. What I'm wondering is to predict which anomalies might appear together in the next data. (I assume that the area is homogeny)

Aim: Generating a dataset having random seismic interpretations

the idea: Generating all kind of combination with the items in the list, and choosing randomly samples in it. Each of elements represents one interpretation information in new created list

In [1]:

```
# function for show the image

def add_pic(image_list, x = 20, y = 15, rows = 1, path=""):

    import matplotlib.image as mpimg
    import matplotlib.pyplot as plt

    fig = plt.figure(figsize=(x, y))

    rows = rows
    columns = len(image_list)
    path = path

    for num, name in enumerate(image_list):

        fig.add_subplot(rows, columns, num+1)

        plt.imshow(mpimg.imread(path+name))

        plt.axis('off')

        plt.title(name)
```

---

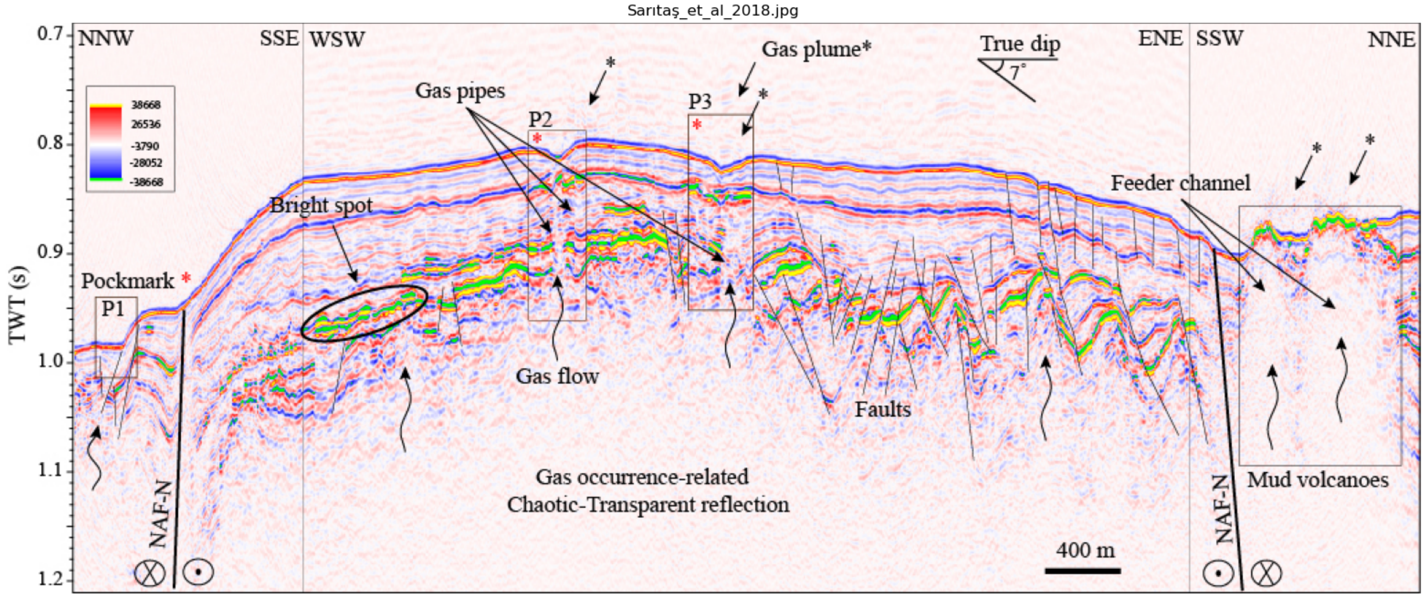
one of the Interpreted marine seismic views acquired the from Sea of Marmara, Western High [3]

In [2]:

```
img_list = ["Sarıtaş_et_al_2018.jpg"]

add_pic(image_list = img_list, x = 20, y = 15, rows = 1, path="pic/")
```

Sarıtaş_et_al_2018.jpg



definitions

- gas pipe = seismic anomaly indicating the presence of upward gas movement through sediments
- gas flow = gas migration
- gas plue = seismic anomaly indicating presence of gas movement to sea column from sea surface
- pockmark = pit seen on the seafloor due to the presence of gas
- mud volcano = structures that throw out mud breccia in a sudden explosion to surface due to high pressure and comminution
- mud diapir = concave Structures that form mid breccia under surface due to high pressure and comminution
- feeder channel = mud volcano breccia is extruded from one major funnel called feeder channel
- falut = a crack in the earth's surface where the rock has divided into two parts that move against each other
- anticline = an anticline is a type of fold that is an arch-like shape and has its oldest beds at its core
- gassy sedimet = sediment with gas bubbles

Example for Marine Science

In [3]:

```
import numpy as np
import pandas as pd
from itertools import combinations, chain
import matplotlib.pyplot as plt
import random
```

list of Seismic Interpretation

In [4]:

```
# write your items as a list

list_of_seismic = ["gassy sediment","mud volcano","mud diapir","feeder channel", "fault","gas flow", "gas plume",
                   "pockmark", "carbonate structure", "gas acummulation","anticline","monocline","channel","erosional surface"]
```

In [5]:

```
len(list_of_seismic)
```

Out[5]:

```
14
```

Combination of items in the list

- "combination" method : itertools.combinations() provides us with all the possible tuples a sequence or set of numbers or letters used in the iterator and the elements are assumed to be unique on the basis of their positions which are distinct for all elements. All these combinations are emitted in lexicographical order. This function takes ‘r’ as input here ‘r’ represents the size of different combinations that are possible. All the combinations emitted are of length ‘r’ and ‘r’ is a necessary argument here. combinations(iterator, r) [4]

In [6]:

```
# 1 to 4 combinations of items list.
# 4: max number of association up to your scenario
# output is nested list with set type items

comb_list = [[b  for b in combinations(list_of_seismic , r)] for r in range(1,4)]

# examples
for i in range(len(comb_list)):

    print(comb_list[i][0:5])
```

```
[('gassy sediment',), ('mud volcano',), ('mud diapir',), ('feeder channel',), ('fault',)]
[('gassy sediment', 'mud volcano'), ('gassy sediment', 'mud diapir'), ('gassy sediment', 'feeder channel'), ('gassy sediment', 'fault'),
('gassy sediment', 'gas flow')]
[('gassy sediment', 'mud volcano', 'mud diapir'), ('gassy sediment', 'mud volcano', 'feeder channel'), ('gassy sediment', 'mud volcano',
'fault'), ('gassy sediment', 'mud volcano', 'gas flow'), ('gassy sediment', 'mud volcano', 'gas plume')]
```

Combining lists in a list

- chain.from_iterable: This function takes a single iterable as an argument and all the elements of the input iterable should also be iterable and it returns a flattened iterable containing all the elements of the input iterable[5]

In [7]:

```
comb_list_sum = list(chain.from_iterable(comb_list))

# lenght of the list

print("lentg of the list: ",len(comb_list_sum))
print("samples from the list: ",comb_list_sum[:20])
```

```
lentg of the list:  469
samples from the list:  [('gassy sediment',), ('mud volcano',), ('mud diapir',), ('feeder channel',), ('fault',), ('gas flow',), ('gas p
lume',), ('pockmark',), ('carbonate structure',), ('gas acummulation',), ('anticline',), ('monocline',), ('channel',), ('erosional surfa
ce',), ('gassy sediment', 'mud volcano'), ('gassy sediment', 'mud diapir'), ('gassy sediment', 'feeder channel'), ('gassy sediment', 'fa
ult'), ('gassy sediment', 'gas flow'), ('gassy sediment', 'gas plume')]
```

Random Samples

In [8]:
```
# getting random samples from the combined list to make more realistic dataset.
# in this example 100 samples used, you can change this number
# if you want to get always the same output in each time, use random.seed(put any number)

random.seed(35)
comb_random = random.sample(comb_list_sum, 100)
```

In [9]:
```
comb_random
print("lentg of the list: ",len(comb_random))
print("samples from the list: ",comb_random[::10])
```

```
lentg of the list:  100
samples from the list:  [('mud diapir', 'gas plume', 'monocline'), ('gassy sediment', 'feeder channel', 'gas flow'), ('mud volcano', 'mu
d diapir', 'pockmark'), ('pockmark', 'anticline', 'monocline'), ('monocline',), ('mud volcano', 'mud diapir', 'fault'), ('mud volcano',
'gas plume', 'monocline'), ('gas flow', 'pockmark', 'anticline'), ('mud volcano', 'gas acummulation', 'erosional surface'), ('gassy sedi
ment', 'feeder channel', 'carbonate structure')]
```

List to Tidy Dataframe for Apriori

- to apply Apriori the dataset should be created with special format including 0 and 1
- The dataset variables will be the "list_of_seismic" we created at the beginning
- the index will be "comb_random" list including random combination samples
- value of the varibales in the dataset will be 1 or 0 according to if the variable is in index or not
- we want the dataset to be as below

In [10]:
```
# Description                        fault    pockmark   gas flow ...........
# products
# ('mud volcano', 'fault')             1         0         1
# ('pockmark', 'fault')                1         1         0
# ('gas flow', 'pockmark')             0         1         1
# .......
```

Create Dataframe

In [11]:
```
# create empty dataframe, add products variable, set this variable as index

df = pd.DataFrame()

df["products"] = comb_random

df = df.set_index("products")
```

In [12]:
```
df.head()
```

Out[12]:

products
(mud diapir, gas plume, monocline)
(gassy sediment, carbonate structure, channel)
(fault, channel, erosional surface)
(fault, channel)
(fault, monocline, channel)

Add New Variables with 0

In [13]:
```
# Adding as many empty variables as the number of items

for i in range(len(list_of_seismic)):

    df[i] = 0
```

In [14]:
```
# change the colums name

df.columns = list_of_seismic
```

In [15]:
```
df.head()
```

Out[15]:

products	gassy sediment	mud volcano	mud diapir	feeder channel	fault	gas flow	gas plume	pockmark	carbonate structure	gas acummulation	anticline	monocline	channel	erosional surface
(mud diapir, gas plume, monocline)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(gassy sediment, carbonate structure, channel)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(fault, channel, erosional surface)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(fault, channel)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(fault, monocline, channel)	0	0	0	0	0	0	0	0	0	0	0	0	0	0

test if one variable is in the index , True False

In [16]:

```
# Example of how to check one variable in products (True or False)
# go to 4.index, 1 first column is whether "Bridge" in (Bridge Pin, Nut, Pick, Tuners)

df_test = df.copy()

df_test.iloc[4,1] = df_test.columns[1] in df_test.index[4]

df_test.iloc[[4]]
```

Out[16]:

products	gassy sediment	mud volcano	mud diapir	feeder channel	fault	gas flow	gas plume	pockmark	carbonate structure	gas acummulation	anticline	monocline	channel	erosional surface
(fault, monocline, channel)	0	False	0	0	0	0	0	0	0	0	0	0	0	0

True False Transformation

In [17]:

```
# Creating loop for checking the variables are in the index or not

for i in range(len(list_of_seismic)):

    for ii in range(len(df)):

        df.iloc[ii,i] = df.columns[i] in df.index[ii]
```

In [18]:

```
df.head()
```

Out[18]:

products	gassy sediment	mud volcano	mud diapir	feeder channel	fault	gas flow	gas plume	pockmark	carbonate structure	gas acummulation	anticline	monocline	channel	erosional surface
(mud diapir, gas plume, monocline)	False	False	True	False	False	False	True	False	False	False	False	True	False	False
(gassy sediment, carbonate structure, channel)	True	False	False	False	False	False	False	False	True	False	False	False	True	False
(fault, channel, erosional surface)	False	False	False	False	True	False	False	False	False	False	False	False	True	True
(fault, channel)	False	False	False	False	True	False	False	False	False	False	False	False	True	False
(fault, monocline, channel)	False	False	False	False	True	False	False	False	False	False	False	True	True	False

True-False to 1-0 Transformation

In [19]:

```
df = df.astype(int)
df.head()
```

Out[19]:

products	gassy sediment	mud volcano	mud diapir	feeder channel	fault	gas flow	gas plume	pockmark	carbonate structure	gas acummulation	anticline	monocline	channel	erosional surface
(mud diapir, gas plume, monocline)	0	0	1	0	0	0	1	0	0	0	0	1	0	0
(gassy sediment, carbonate structure, channel)	1	0	0	0	0	0	0	0	1	0	0	0	1	0
(fault, channel, erosional surface)	0	0	0	0	1	0	0	0	0	0	0	0	1	1
(fault, channel)	0	0	0	0	1	0	0	0	0	0	0	0	1	0
(fault, monocline, channel)	0	0	0	0	1	0	0	0	0	0	0	1	1	0

Graph of Items Count in Dataset

In [20]:

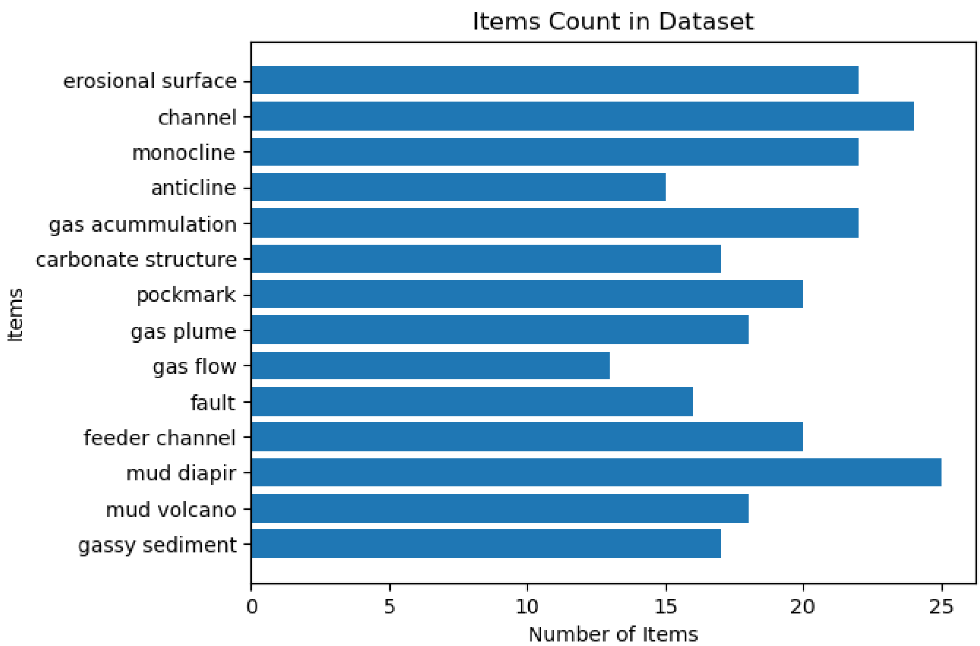
```
plt.barh(df.columns, df.sum())

plt.title("Items Count in Dataset")

plt.xlabel("Number of Items")

plt.ylabel("Items")

plt.show();
```



Save as csv

In [19]:

```
df.to_csv("seismic_apriori_analysis.csv")
```

Create Dataset by Function

In [21]:

```
import numpy as np
import pandas as pd
from itertools import combinations, chain
import matplotlib.pyplot as plt
import random
```

In [22]:

```
def create_apriori_dataset(item_list, n_comb=1, sample_value = 2, transformation= True, seed = True, seed_number = 35,
                           save=True, dataset_name ="apriori_analysis" ):


    """
```

```
    Parameters:
        - item_list = list of objects you want to analysis
        - n_comb = length of the iterable. Default value is 1.
        - sample_value = sample value from the population. Default value is 2
        - transformation = Boolean. If True, transform data to 0 and 1. If False, the dataset variables'value are "True" and False".
        - seed = use random.seed() to get always the same result. Default is True. Boolean
        - seed_number = Default value is 35. Change what ever you want.
        - save = Save the dataset as csv. Default is True.Boolean
        - dataset_name = output name of the dataset. Default name is "apriori_analysis"

    info:
        - if you get "ValueError: Sample larger than population or is negative" it is mean your sample_value bigger than number of
        observations, In this case you can incerase the n_comb value or degrease the sample_value

    Return:

        - dataframe

    Example:
        basket_list = [flour, sugar,tea,bread,pasta,buttermilk,cheese,chocolate,detergent,toothpaste]

        df_marine = create_apriori_dataset(item_list = basket_list, n_comb=4, sample_value = 100,
                                             save = True, dataset_name = "market")

    Info:
        to calculate max sample value for creating dataset according to given n_comb:

        Ex:
        n_comb = 3
        len of item list = 8
        max sample_value = (8/1) + (8*7/2*1) + (8*7*6/3*2*1)  = 92

    """

    # create combination
    comb_list = [[b  for b in combinations(item_list, r)] for r in range(1 , n_comb+1)]

    # combining lists in the list
    comb_list_sum = list(chain.from_iterable(comb_list))

    # use seed
    if seed:

        random.seed(seed_number)


    comb_random = random.sample(comb_list_sum, sample_value)

    # create dataframe
    df_new = pd.DataFrame()

    df_new["products"] = comb_random

    df_new.set_index("products", inplace = True)

    # make all 0
    for i in range(len(item_list)):

        df_new[i] = 0

    # give columns name
    df_new.columns = item_list

    # True and False if the variable in the index
    for i in range(len(item_list)):

        for ii in range(len(df_new)):

            df_new.iloc[ii,i] = df_new.columns[i] in df_new.index[ii]


    # transaformation true-flase to 0-1
    if transformation:

        df_new = df_new.astype(int)


    # save
    if save:

        df_new.to_csv(dataset_name+".csv")

    return df_new
```

In [23]:

```
list_of_seismic = ["gassy sediment","mud volcano","mud diapir","feeder channel", "fault","gas flow", "gas plume",
                   "pockmark", "carbonate structure", "gas acummulation","anticline","monocline","channel","erosional surface"]
```

In [24]:

```
df_marine = create_apriori_dataset(item_list = list_of_seismic,
```

```python
                                         n_comb=4,
                                         sample_value =92,
                                         transformation= True,
                                         save = False)
```

In [25]:

```python
df_marine.head()
```

Out[25]:

products	gassy sediment	mud volcano	mud diapir	feeder channel	fault	gas flow	gas plume	pockmark	carbonate structure	gas acummulation	anticline	monocline	channel	erosional surface
(mud diapir, carbonate structure, gas acummulation, erosional surface)	0	0	1	0	0	0	0	0	1	1	0	0	0	1
(gassy sediment, gas flow, carbonate structure, monocline)	1	0	0	0	0	1	0	0	1	0	0	1	0	0
(mud diapir, gas flow, pockmark)	0	0	1	0	0	1	0	1	0	0	0	0	0	0
(gassy sediment, gas flow, monocline, erosional surface)	1	0	0	0	0	1	0	0	0	0	0	1	0	1
(feeder channel, gas flow, carbonate structure)	0	0	0	1	0	1	0	0	1	0	0	0	0	0

References

[1] - Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.

[2] - Wikipedia. https://en.wikipedia.org/wiki/Apriori_algorithm

[3] - Sarıtaş, H. (2018). Gas occurrence and shallow conduit systems in the Western Sea of Marmara: A review and new acoustic evidence. Geo-Marine Letters, 38(5), 385-432. https://doi.org/10.1007/s00367-018-0547-5

[4] - itertools.combinations. https://www.geeksforgeeks.org/python-itertools-combinations-function/

[5] - itertools.chain. https://docs.python.org/3/library/itertools.html

In [ ]:

Association Rule Learning (Apriori Analysis) for Marine seismic Interpretation

- The Apriori algorithm even called Basket Analysis is generally applied for selling more items in supermarkets.
- However; if you want, you can use this algorithm for different topics in which you want to study on frequency.
- The Question to be Answered is "which item combinations are frequently seem together".
- In this study, I showed two ways of application of apriori algorithm.
  - Firstly, I used manual functions created by me,
  - Secondly, I used the apriori function from the mlxted library.
- Using the mlxtend.apriori function is much easier and gives more detailed results. But the two applications offer the same results.

Three Main Metrics

- **Support**:
  - Support(X,Y) = Freq(X,Y) / N
  - Probability of X and y occurring together = Freq of X and Y occurring together / Numb of All Transactions
  - If support value is high, it means that these pair of items are bought together much more
  - The number of transactions in which a specific product (or combination of products) occurs
  - if one item are observed below the threshold value, it means that is not useful for the user
- **Confidence**
  - Confidence(X,Y) = Freq(X,Y) / Freq(X)
  - Probability of buying Y when X is bought = Freq of X and Y occurring together/ Freq of X
  - example 25% means that the association occurs 25% percentage of the case
- **Lift**
  - Lift = Support(X,Y) / (Support(X)*Support(Y))
  - When X is purchased, the probability of Y being bought increases by the lift value
  - performance metric
  - lift of a rule is performance metric that indicates the strength of the association between the products in the rule[1]
  - If the lift of a rule is higher than 1, the lift value tells you how strongly the righthand side product depends on the left-hand side[1].

1 - Apriori Application by Manual

In [1]:
```
import numpy as np
import pandas as pd
from itertools import combinations, chain
import random
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv("seismic_apriori_analysis.csv", index_col=[0])
```

In [3]:
```
df.head(2)
```

Out[3]:

products	gassy sediment	mud volcano	mud diapir	feeder channel	fault	gas flow	gas plume	pockmark	carbonate structure	gas acummulation	anticline	monocline	channel	erosional surface
('mud volcano', 'gas flow', 'gas plume', 'gas acummulation')	0	1	0	0	0	1	1	0	0	1	0	0	0	0
('gassy sediment', 'mud volcano', 'gas plume', 'anticline')	1	1	0	0	0	0	1	0	0	0	1	0	0	0

Step 1. Computing the support for each individual item

- Support value is the number of unique items in the all interpretations
- It is calculated by summing the row values of all variables in the data set separately.
- Applied the Support threshold to find out which products are observed more
- Add Confidence and Lift constant values to concat dataframe at the end without problem

---

Graph of Items Count in Dataset

In [4]:

```
# frequency of the unique items
df.sum()
```

Out[4]:

```
gassy sediment          27
mud volcano             27
mud diapir              25
feeder channel          26
fault                   26
gas flow                31
gas plume               21
pockmark                31
carbonate structure     19
gas acummulation        20
anticline               21
monocline               20
channel                 21
erosional surface       26
dtype: int64
```

In [5]:

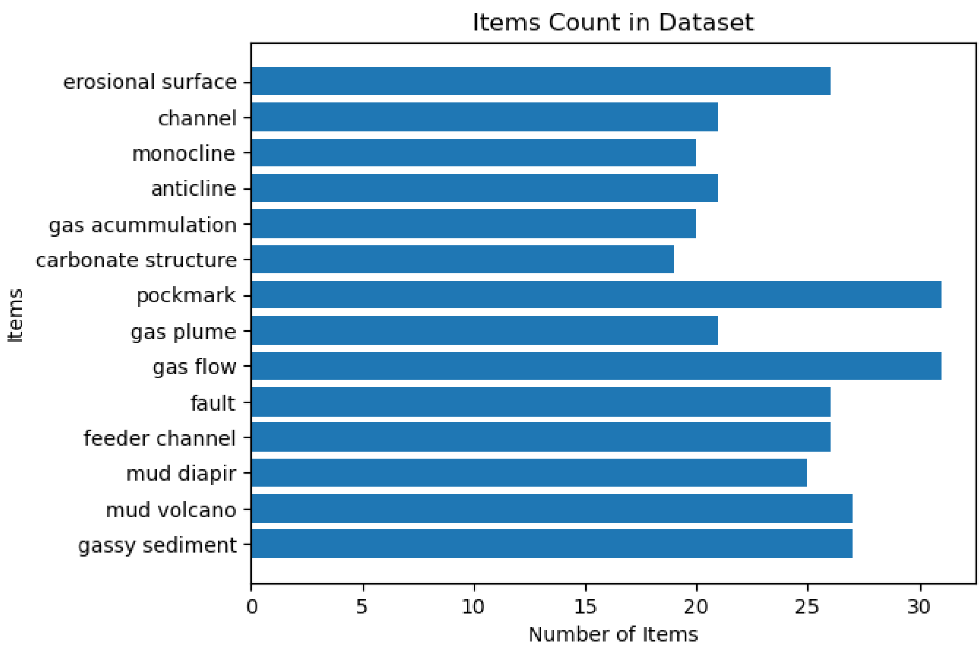
```
plt.barh(df.columns, df.sum())

plt.title("Items Count in Dataset")

plt.xlabel("Number of Items")

plt.ylabel("Items")

plt.show();
```



Support Value Graph

In [5]:

```
# support value

round(df.sum() / df.shape[0],3)
```

Out[5]:

```
gassy sediment          0.293
mud volcano             0.293
mud diapir              0.272
feeder channel          0.283
fault                   0.283
gas flow                0.337
gas plume               0.228
pockmark                0.337
carbonate structure     0.207
gas acummulation        0.217
```

```
anticline            0.228
monocline            0.217
channel              0.228
erosional surface    0.283
dtype: float64
```

In [6]:

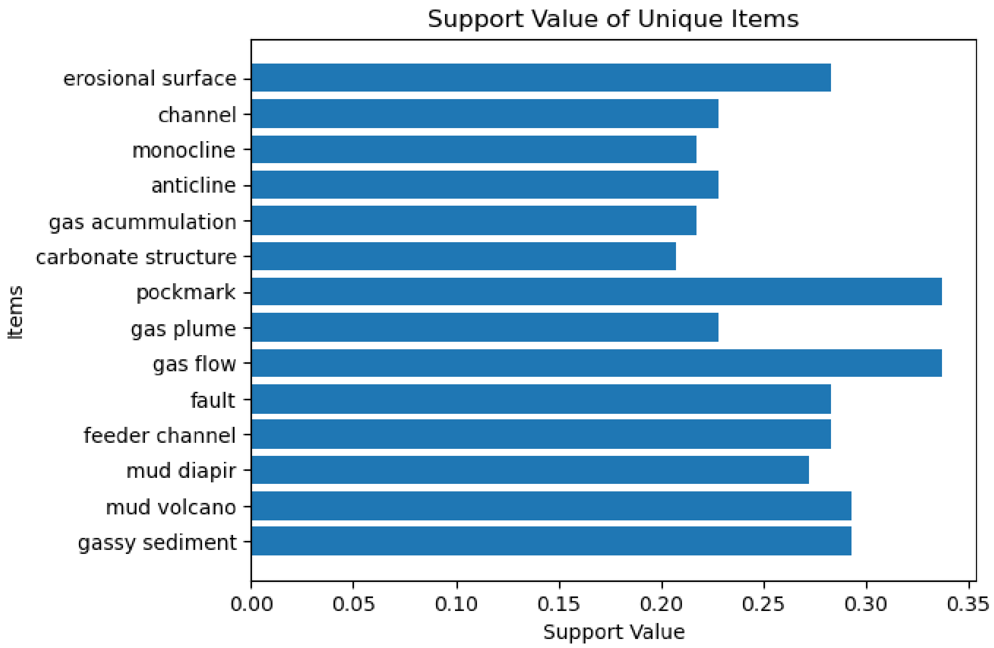
```python
plt.barh(df.columns, round(df.sum() / df.shape[0],3))

plt.title("Support Value of Unique Items")

plt.xlabel("Support Value")

plt.ylabel("Items")

plt.show();
```



Function for unique item

In [6]:

```python
def one_apriori(dataframe, Support=False, support_value = 0.01):

    """
    Parameters:
        dataframe : initial dataframe for the apriori analysis.
        Support: If True apply Support Threshold to eliminate below products. Default value is False
        support_value: support threshold value. Default value  = 0.2

    Info:
        This function calculate freq and support values of the each individual item in the transactions.

    Return:
        df_new : Dataframe
    """

    # frequency
    freq = dataframe.sum()

    #support value
    support_values = freq.agg(lambda x: round(x / dataframe.shape[0], 3))

    #dataframe
    df_new = pd.DataFrame(support_values, columns = ["Support"])

    df_new = df_new.sort_values("Support", ascending = False)

    df_new["Freq"] = freq

    # Constant Confidence anf Lift values

    df_new["Confidence"] = 1

    df_new["Lift"] = np.NaN

    # Apply Support Threshold

    if Support:

        df_new = df_new.loc[(df_new["Support"] >= support_value)]
```

```
    return df_new
```

In [7]:

```
df_one = one_apriori(dataframe = df)
df_one
```

Out[7]:

	Support	Freq	Confidence	Lift
gas flow	0.337	31	1	NaN
pockmark	0.337	31	1	NaN
gassy sediment	0.293	27	1	NaN
mud volcano	0.293	27	1	NaN
feeder channel	0.283	26	1	NaN
fault	0.283	26	1	NaN
erosional surface	0.283	26	1	NaN
mud diapir	0.272	25	1	NaN
gas plume	0.228	21	1	NaN
anticline	0.228	21	1	NaN
channel	0.228	21	1	NaN
gas acummulation	0.217	20	1	NaN
monocline	0.217	20	1	NaN
carbonate structure	0.207	19	1	NaN

Step 2. Computation the support value for item pairs

- The support value is the number of coexistence of product pairs in each interpretaion.
- Add Confidence and Lift constant values are calculated

In [8]:

```
# combinations of two items

comb_number = 2

two_list= [list(i) for i in combinations(df_one.index, comb_number)]

two_list[0:10]
```

Out[8]:

```
[['gas flow', 'pockmark'],
 ['gas flow', 'gassy sediment'],
 ['gas flow', 'mud volcano'],
 ['gas flow', 'feeder channel'],
 ['gas flow', 'fault'],
 ['gas flow', 'erosional surface'],
 ['gas flow', 'mud diapir'],
 ['gas flow', 'gas plume'],
 ['gas flow', 'anticline'],
 ['gas flow', 'channel']]
```

In [9]:

```
# example for ('mud diapir', 'gas plume')

df[['mud diapir', 'gas plume']]
```

Out[9]:

	mud diapir	gas plume
products		
('mud volcano', 'gas flow', 'gas plume', 'gas acummulation')	0	1
('gassy sediment', 'mud volcano', 'gas plume', 'anticline')	0	1
('feeder channel', 'gas flow', 'gas acummulation', 'erosional surface')	0	0
('mud diapir', 'gas plume', 'pockmark', 'gas acummulation')	1	1
('mud volcano', 'fault', 'erosional surface')	0	0
...	...	...
('mud diapir', 'fault', 'gas flow', 'channel')	1	0
('mud volcano', 'feeder channel', 'fault', 'carbonate structure')	0	0
('gassy sediment', 'feeder channel', 'pockmark', 'carbonate structure')	0	0
('feeder channel', 'gas flow', 'pockmark', 'channel')	0	0
('mud volcano', 'mud diapir', 'pockmark', 'gas acummulation')	1	0

92 rows × 2 columns

In [10]:

```
# if 'mud diapir'and 'gas plume' exist in index, sum of items should be 2

df[df[['mud diapir', 'gas plume']].sum(axis=1) == 2]
```

Out[10]:

products	gassy sediment	mud volcano	mud diapir	feeder channel	fault	gas flow	gas plume	pockmark	carbonate structure	gas acummulation	anticline	monocline	channel	erosional surface
('mud diapir', 'gas plume', 'pockmark', 'gas acummulation')	0	0	1	0	0	0	1	1	0	1	0	0	0	0
('mud diapir', 'feeder channel', 'gas flow', 'gas plume')	0	0	1	1	0	1	1	0	0	0	0	0	0	0
('mud volcano', 'mud diapir', 'feeder channel', 'gas plume')	0	1	1	1	0	0	1	0	0	0	0	0	0	0
('mud diapir', 'gas plume', 'pockmark', 'carbonate structure')	0	0	1	0	0	0	1	1	1	0	0	0	0	0

In [11]:

```
# support value and frequency

values_ = []

freq = []

for i in range(len(two_list)):

    items = df[df[two_list[i]].sum(axis=1) == len(two_list[i])]

    item_support = len(items) / df.shape[0]

    values_.append(item_support)

    freq.append(len(items))
```

In [12]:

```
# create dataframe

df_test = pd.DataFrame(values_, columns=["Support"])

df_test["products"] = two_list

df_test["Freq"] = freq

df_test = df_test.set_index("products")

df_test =df_test.sort_values(by = "Support", ascending=False)
```

In [13]:

```
df_test.head()
```

Out[13]:

products	Support	Freq
[pockmark, feeder channel]	0.108696	10
[gas flow, mud volcano]	0.108696	10
[gassy sediment, erosional surface]	0.097826	9
[gassy sediment, mud volcano]	0.097826	9
[mud volcano, fault]	0.097826	9

In [14]:

```
list_conf = []

for i in range(len(df_test)):

    conf = df_test.Freq[i] / df_one.loc[[df_test.index[i][0]]]["Freq"]

    list_conf.append(np.round(conf[0],2))

df_test["confidence"] = list_conf
```

In [15]:
```
# calculate Lift

list_lift = []

for i in range(len(df_test)):

    lift = df_test.Support[i] / (df_one.loc[[df_test.index[i][0]]].Support[0] * df_one.loc[[df_test.index[i][1]]].Support[0])

    list_lift.append(np.round(lift,2))


df_test["lift"] = list_lift
```

In [16]:
```
# support value bigger than 0.06

df_test = df_test.loc[df_test["Support"] >= 0.06 ]

df_test.head()
```

Out[16]:

products	Support	Freq	confidence	lift
[pockmark, feeder channel]	0.108696	10	0.32	1.14
[gas flow, mud volcano]	0.108696	10	0.32	1.10
[gassy sediment, erosional surface]	0.097826	9	0.33	1.18
[gassy sediment, mud volcano]	0.097826	9	0.33	1.14
[mud volcano, fault]	0.097826	9	0.33	1.18

Function for two items

In [17]:
```
def two_apriori(dataframe , dataframe_1 , Support = False, support_value = 0.01 ):

    """
    Parameters:
        dataframe : initial dataframe for the apriori analysis
        dataframe_1 : first step output dataframe which contains individual products
        Support: If "True", apply Support Threshold to eliminate products under this value. Default boolean value is False
        support_value: support threshold value. Default value  = 0.2

    Info:
        This function calculates the frequency and support values of coexistence product pairs in each transaction.

    Return:
        df_new : Dataframe
    """

    # combination of two items
    item_list= [list(i) for i in combinations(dataframe_1.index, 2)]


    # FIND ITEMS MATCHED WITH COMBINATION
    values_ = []
    freq = []

    for i in range(len(item_list)):

    # try:  if the combination has not any items with given support value, dont break
        try:

            items = dataframe[dataframe[item_list[i]].sum(axis=1) == len(item_list[i])]

            item_support = round(len(items) / dataframe.shape[0],3)

            values_.append(item_support)

            freq.append(len(items))

        except (KeyError) as Error :

            print("No items:",Error)


    # DATAFRAME with FREQ and SUPPORT VARIABLES

    # support value
    df_new = pd.DataFrame(values_, columns=["Support"])

    # item frequency
    df_new["Freq"] = freq

    # to set index add items to dataframe
    df_new["products"] = item_list
```

```
    df_new = df_new.set_index("products")

    # sort
    df_new = df_new.sort_values(by = "Support", ascending=False)


    # CONFIDENCE
    list_conf = []
    # iterate in each index in df_new
    for i in range(len(df_new)):


        conf = df_new.Freq[i] / dataframe_1.loc[[df_new.index[i][0]]]["Freq"]

        list_conf.append(np.round(conf[0],2))

    df_new["Confidence"] = list_conf


    # LIFT
    list_lift = []
    for i in range(len(df_new)):

        lift = df_new.Support[i] / (dataframe_1.loc[[df_new.index[i][0]]].Support[0] * dataframe_1.loc[[df_new.index[i][1]]].Support[0])

        list_lift.append(np.round(lift,3))


    df_new["Lift"] = list_lift


    # SUPPORT THRESHOLD

    if Support:

        df_new = df_new.loc[df_new["Support"] >= support_value ]

    return df_new
```

In [18]:
```
df_two = two_apriori(dataframe = df,
                     dataframe_1 = df_one,
                     Support = True)
```

In [19]:
```
df_two.head()
```

Out[19]:

products	Support	Freq	Confidence	Lift
[pockmark, feeder channel]	0.109	10	0.32	1.143
[gas flow, mud volcano]	0.109	10	0.32	1.104
[gassy sediment, erosional surface]	0.098	9	0.33	1.182
[gassy sediment, mud volcano]	0.098	9	0.33	1.142
[mud volcano, fault]	0.098	9	0.33	1.182

Step 3. Computation the support value for item combinations (more than two)

- The support value is the number of coexistence of product combinations in each interpretaion.
- Add Confidence and Lift constant values are calculated

In [20]:
```
# combination of the pair items

comb_number = 2

three_list = [i for i in combinations(df_two.index, 2)]

three_list[0:10]
```

Out[20]:
```
[(['pockmark', 'feeder channel'], ['gas flow', 'mud volcano']),
 (['pockmark', 'feeder channel'], ['gassy sediment', 'erosional surface']),
 (['pockmark', 'feeder channel'], ['gassy sediment', 'mud volcano']),
 (['pockmark', 'feeder channel'], ['mud volcano', 'fault']),
 (['pockmark', 'feeder channel'], ['pockmark', 'mud diapir']),
 (['pockmark', 'feeder channel'], ['pockmark', 'erosional surface']),
 (['pockmark', 'feeder channel'], ['gas flow', 'fault']),
 (['pockmark', 'feeder channel'], ['gas flow', 'mud diapir']),
```

```
([ 'pockmark', 'feeder channel'], ['gassy sediment', 'fault']),
([ 'pockmark', 'feeder channel'], ['fault', 'anticline'])]
```

In [21]:

```python
# to get unique combinations of the items

three_list_set = []

for i in range(len(three_list)):

    chain_ = set(chain.from_iterable(three_list[i]))

    if chain_ not in three_list_set:

        three_list_set.append(chain_)
```

In [22]:

```python
# set to list

set_to_list = [list(three_list_set[i]) for i in range(len(three_list_set))]
```

In [23]:

```python
set_to_list[0:10]
```

Out[23]:

```
[['pockmark', 'feeder channel', 'mud volcano', 'gas flow'],
 ['erosional surface', 'pockmark', 'feeder channel', 'gassy sediment'],
 ['pockmark', 'feeder channel', 'mud volcano', 'gassy sediment'],
 ['pockmark', 'fault', 'feeder channel', 'mud volcano'],
 ['pockmark', 'feeder channel', 'mud diapir'],
 ['erosional surface', 'pockmark', 'feeder channel'],
 ['pockmark', 'feeder channel', 'fault', 'gas flow'],
 ['pockmark', 'feeder channel', 'mud diapir', 'gas flow'],
 ['pockmark', 'feeder channel', 'fault', 'gassy sediment'],
 ['anticline', 'pockmark', 'feeder channel', 'fault']]
```

In [24]:

```python
# support values

values_ = []

freq= []

for i in range(len(set_to_list)):

    items = df[df[set_to_list[i]].sum(axis=1) == len(set_to_list[i])]

    item_support = len(items) / df.shape[0]

    values_.append(item_support)

    freq.append(len(items))
```

In [25]:

```python
# dataframe

df_test = pd.DataFrame(values_, columns=["Support"])

df_test["products"] = set_to_list

df_test["Freq"] = freq

df_test = df_test.set_index("products")

df_test= df_test.sort_values(by = "Support", ascending=False)
```

In [26]:

```python
df_test.head()
```

Out[26]:

	Support	Freq
products		
[gas acummulation, erosional surface, gas flow]	0.032609	3
[channel, monocline, gas plume]	0.032609	3
[mud diapir, mud volcano, gas flow]	0.032609	3
[anticline, gas plume, fault]	0.032609	3
[channel, erosional surface, carbonate structure]	0.032609	3

In [27]:

```python
# Confidence

list_conf= []

for i in range(len(df_test)):

    conf = df_test.Freq[i] / df_one.loc[[df_test.index[i][0]]]["Freq"]
```

```
    list_conf.append(np.round(conf[0],2))

df_test["Confidence"] = list_conf
```

In [28]:
```
df_test.head()
```

Out[28]:

products	Support	Freq	Confidence
[gas acummulation, erosional surface, gas flow]	0.032609	3	0.15
[channel, monocline, gas plume]	0.032609	3	0.14
[mud diapir, mud volcano, gas flow]	0.032609	3	0.12
[anticline, gas plume, fault]	0.032609	3	0.14
[channel, erosional surface, carbonate structure]	0.032609	3	0.14

In [29]:
```
# lift

list_support_multiple = []

list_lift= []

for i in range(len(df_test)):

    list_support = []

    for ii in range(len(df_test.index[i])):

        list_support.append(df_one.loc[[df_test.index[i][ii]]].Support[0])

    list_support_multiple.append(np.round(np.prod(list_support),3))

    lift = df_test.Support[i] / list_support_multiple[i]

    list_lift.append(np.round(lift,3))

df_test["Lift"] = list_lift
```

In [30]:
```
df_test = df_test.loc[df_test.Support >= 0.01]
df_test.head()
```

Out[30]:

products	Support	Freq	Confidence	Lift
[gas acummulation, erosional surface, gas flow]	0.032609	3	0.15	1.553
[channel, monocline, gas plume]	0.032609	3	0.14	2.964
[mud diapir, mud volcano, gas flow]	0.032609	3	0.12	1.208
[anticline, gas plume, fault]	0.032609	3	0.14	2.174
[channel, erosional surface, carbonate structure]	0.032609	3	0.14	2.508

Function for three and more items

In [31]:
```
def more_apriori(dataframe , dataframe_1 , dataframe_2 , Support = False, support_value = 0.01 ):

    """
    Parameters:
        dataframe : initial dataframe for the apriori analysis
        dataframe_1 : first step output dataframe which contains individual products
        dataframe_2 : one of the the other steps output (ex. df_two, df_three,df_four) dataframe which contains combined products
        Support: If "True", apply Support Threshold to eliminate products under this value. Default boolean value is False
        support_value: support threshold value. Default value  = 0.2

    Info:
        - This function calculates the frequency and support values of coexistence product combinations(more than two)
        in each transaction.
        - one_apriori and two_apriori functions should be done before this function.


    Return:
        df_new : Dataframe

    Ex:
        df_three = more_apriori(df, df_one, df_two, Support = True, support_value = 0.05)
    Ex:
        df_four = more_apriori(df, df_one, df_three)
    """

    # finding probable combination of the pairs
```

```python
item_list = [i for i in combinations(dataframe_2.index, 2)]

# to get unique combinations of the items
item_list_set = []

for i in range(len(item_list)):

    chain_ = set(chain.from_iterable(item_list[i]))

    if chain_ not in item_list_set:

        item_list_set.append(chain_)

# set to list for easy iteration
set_to_list = [list(item_list_set[i]) for i in range(len(item_list_set))]

# FIND ITEMS MATCHED WITH COMBINATION
values_ = []
freq = []

for i in range(len(set_to_list)):

    try:

        items = dataframe[dataframe[set_to_list[i]].sum(axis=1) == len(set_to_list[i])]

        item_support = round(len(items) / dataframe.shape[0],3)

        values_.append(item_support)

        freq.append(len(items))

    except (KeyError) as Error :

        print("No items:",Error)

# DATAFRAME with FREQ and SUPPORT VARIABLES

df_new = pd.DataFrame(values_, columns=["Support"])

df_new["products"] = set_to_list

df_new["Freq"] = freq

df_new = df_new.set_index("products")

df_new = df_new.sort_values(by = "Support", ascending=False)


# CONFIDENCE

list_conf = []

for i in range(len(df_new)):

    conf = df_new.Freq[i] / dataframe_1.loc[[df_new.index[i][0]]]["Freq"]

    list_conf.append(np.round(conf[0],2))

df_new["Confidence"] = list_conf


# LIFT

list_lift = []
list_support_multiple = []

for i in range(len(df_new)):

    list_support = []

    for ii in range(len(df_new.index[i])):

        list_support.append(dataframe_1.loc[[df_new.index[i][ii]]].Support[0])

    list_support_multiple.append(np.round(np.prod(list_support),3))

    lift = df_new.Support[i] / list_support_multiple[i]

    list_lift.append(np.round(lift,3))


df_new["Lift"] = list_lift


# SUPPORT THRESHOLD

if Support:

    df_new = df_new.loc[df_new["Support"] >= support_value]
```

```
    return df_new
```

In [32]:
```
df_three = more_apriori(dataframe = df,
                        dataframe_1 = df_one,
                        dataframe_2 = df_two,
                        Support = True)
```

In [33]:
```
df_three.head()
```

Out[33]:

products	Support	Freq	Confidence	Lift
[gas acummulation, erosional surface, gas flow]	0.033	3	0.15	1.571
[channel, monocline, gas plume]	0.033	3	0.14	3.000
[mud diapir, mud volcano, gas flow]	0.033	3	0.12	1.222
[anticline, gas plume, fault]	0.033	3	0.14	2.200
[channel, erosional surface, carbonate structure]	0.033	3	0.14	2.538

Combine The Data Frames

In [34]:
```
# function for concat the dataframes and apply support threshold

def combine_apriori(dataframes_list ,Support = False, support_value = 0.01):

    df_new = pd.concat(dataframes)

        # SUPPORT THRESHOLD

    if Support:

        df_new = df_new.loc[df_new["Support"] >= support_value]

    return df_new
```

In [35]:
```
# apply the function

dataframes = [df_two,df_three]

df_combine = combine_apriori(dataframes, Support = True,support_value= 0.085)
```

In [44]:
```
df_combine.sort_values(by ="Support",ascending = False)
```

Out[44]:

products	Support	Freq	Confidence	Lift
[pockmark, feeder channel]	0.109	10	0.32	1.143
[gas flow, mud volcano]	0.109	10	0.32	1.104
[gassy sediment, erosional surface]	0.098	9	0.33	1.182
[gassy sediment, mud volcano]	0.098	9	0.33	1.142
[mud volcano, fault]	0.098	9	0.33	1.182
[pockmark, mud diapir]	0.098	9	0.29	1.069
[pockmark, erosional surface]	0.087	8	0.26	0.912
[gas flow, fault]	0.087	8	0.26	0.912
[gas flow, mud diapir]	0.087	8	0.26	0.949

Interpretation

assocciation rule for the fifth observation

- to interpretaion we can chose of the metrics which are support, confidence or lift)
- Mud volcano and fault appear together in 10% of all interpretaions (ssupport).
- In other words, mud volcano and fault are seen together in 10 out of 100 interpretations.
- 33%(confidence) of interpretations containing mud volcano are included in the fault
- fault presence increases by 1.18 times in interpretations containing mud volcano (lift)

---

2 -Apriori Analysis with mlxtend[2]

In [37]:
```
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
```

In [38]:
```
df = pd.read_csv("seismic_apriori_analysis.csv", index_col=[0])
```

- mlxtend wants the variables to have True and False values, but according to the version of mltextend, values 0 and 1 are also expected to be accepted

In [39]:
```
# transform to boolen values

df = df.astype(bool)
```

In [40]:
```
# apply apriori

df_apriori = apriori(df, min_support = 0.01, use_colnames = True)
```

In [41]:
```
df_apriori
```

Out[41]:

	support	itemsets
0	0.293478	(gassy sediment)
1	0.293478	(mud volcano)
2	0.271739	(mud diapir)
3	0.282609	(feeder channel)
4	0.282609	(fault)
...	...	...
389	0.010870	(gas acummulation, erosional surface, anticlin...
390	0.010870	(gas acummulation, channel, pockmark, gas plume)
391	0.010870	(anticline, pockmark, monocline, carbonate str...
392	0.010870	(channel, erosional surface, pockmark, carbona...
393	0.010870	(gas acummulation, erosional surface, anticlin...

394 rows × 2 columns

In [42]:
```
# apply association rules

df_rules = association_rules(df_apriori, metric = "support", min_threshold = 0.01)
```

In [43]:
```
# sorting

df_rules.sort_values(by ="support",ascending = False).head(10)
```

Out[43]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
33	(gas flow)	(mud volcano)	0.336957	0.293478	0.108696	0.322581	1.099164	0.009806	1.042961
32	(mud volcano)	(gas flow)	0.293478	0.336957	0.108696	0.370370	1.099164	0.009806	1.053069
78	(pockmark)	(feeder channel)	0.336957	0.282609	0.108696	0.322581	1.141439	0.013469	1.059006
79	(feeder channel)	(pockmark)	0.282609	0.336957	0.108696	0.384615	1.141439	0.013469	1.077446
0	(mud volcano)	(gassy sediment)	0.293478	0.293478	0.097826	0.333333	1.135802	0.011697	1.059783
1	(gassy sediment)	(mud volcano)	0.293478	0.293478	0.097826	0.333333	1.135802	0.011697	1.059783
58	(pockmark)	(mud diapir)	0.336957	0.271739	0.097826	0.290323	1.068387	0.006262	1.026186
59	(mud diapir)	(pockmark)	0.271739	0.336957	0.097826	0.360000	1.068387	0.006262	1.036005
31	(mud volcano)	(fault)	0.293478	0.282609	0.097826	0.333333	1.179487	0.014887	1.076087
30	(fault)	(mud volcano)	0.282609	0.293478	0.097826	0.346154	1.179487	0.014887	1.080563

Interpretation

antecedents = first item/items

consequents = second item/items

for the first row assocciation rule

- to interpretaion we can chhose of the metrics which are support, confidence or lift)
- Mud volcano and gas flow appear together in 10% of all interpretaions (ssupport).
- In other words, mud volcano and gas flow are seen together in 10 out of 100 interpretations.
- 44%(confidence) of interpretations containing mud volcano are included in the gas flow
- gas flow presence increases by 1.55 times in interpretations containing mud volcano (lift)

References

[1] - Joos Korstanje. The Apriori algorithm. https://towardsdatascience.com/the-apriori-algorithm-5da3db9aea95

[2] - mlxtend.apriori. http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/

Hakan Sarıtaş

- linkedin: https://www.linkedin.com/in/hakansaritas/
- github : https://github.com/hakansaritas
- email: saritas_hakan@yahoo.com

In [ ]:

```

```