Week – 7

Write a program of Naive Bayesian classification using Python programming language.

AIM: To write a program of Naive Bayesian classification using Python programming language.

DESCRIPTION:

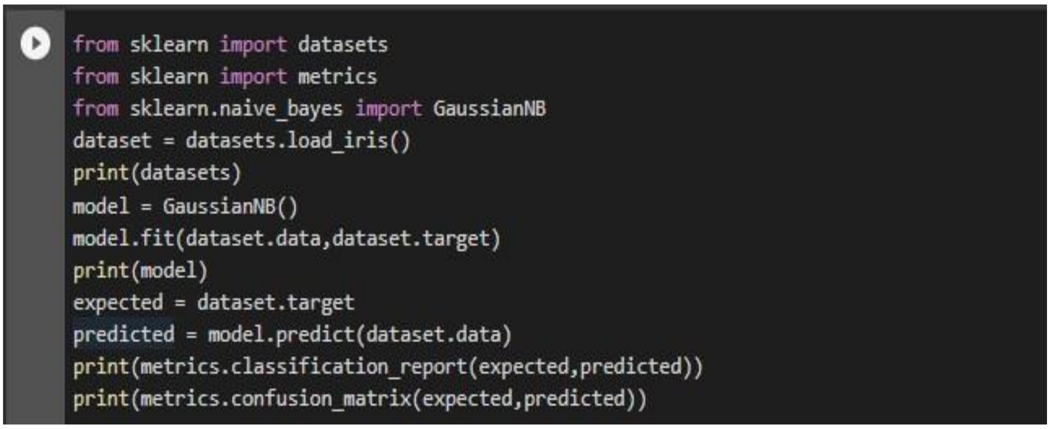
Naive Bayesian is a statistical classification technique based on bayes theorem.

It is fast, accurate and liable algorithm.

Naive Bayesian has classifiers have high accuracy and speed on large datasets.

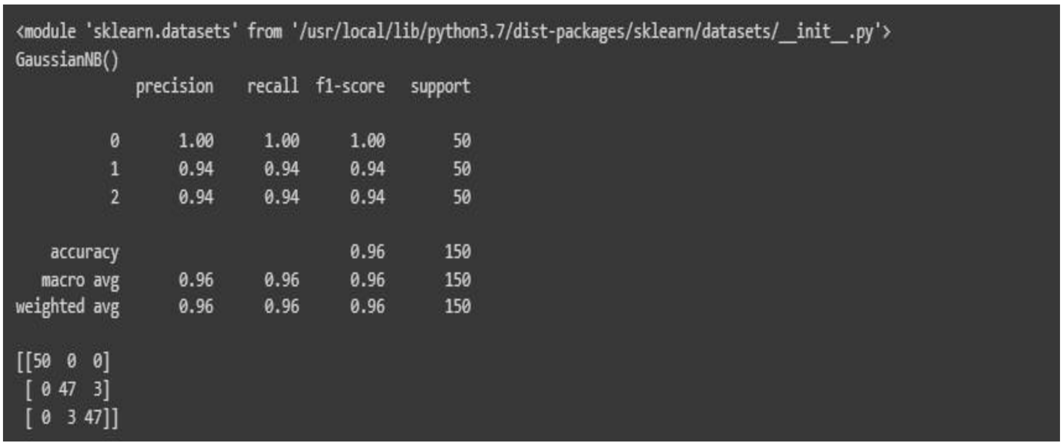
The Naive Bayes classification algorithm is **a probabilistic classifier**. It is based on probability models that incorporate strong independence assumptions.

PROGRAM:



```
from sklearn import datasets
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
dataset = datasets.load_iris()
print(datasets)
model = GaussianNB()
model.fit(dataset.data,dataset.target)
print(model)
expected = dataset.target
predicted = model.predict(dataset.data)
print(metrics.classification_report(expected,predicted))
print(metrics.confusion_matrix(expected,predicted))
```

OUTPUT:



```
<module 'sklearn.datasets' from '/usr/local/lib/python3.7/dist-packages/sklearn/datasets/__init__.py'>
GaussianNB()
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       0.94      0.94      0.94        50
           2       0.94      0.94      0.94        50

    accuracy                           0.96       150
   macro avg       0.96      0.96      0.96       150
weighted avg       0.96      0.96      0.96       150

[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
```

Week – 10

Write a Python program to generate frequent item sets / association rules using Apriori algorithm.

AIM: To write a Python program to generate frequent item sets / association rules using Apriori algorithm.

DECRIPTION:

The algorithm is used to finding frequent itemset in a dataset for boolean association rule.

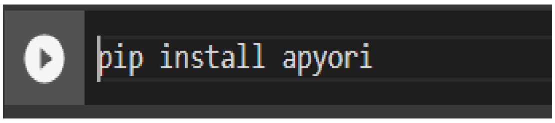
Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

To improve the efficiency of level-wise generation of frequent itemsets, an important property is used called **Apriori property** which helps by reducing the search space.

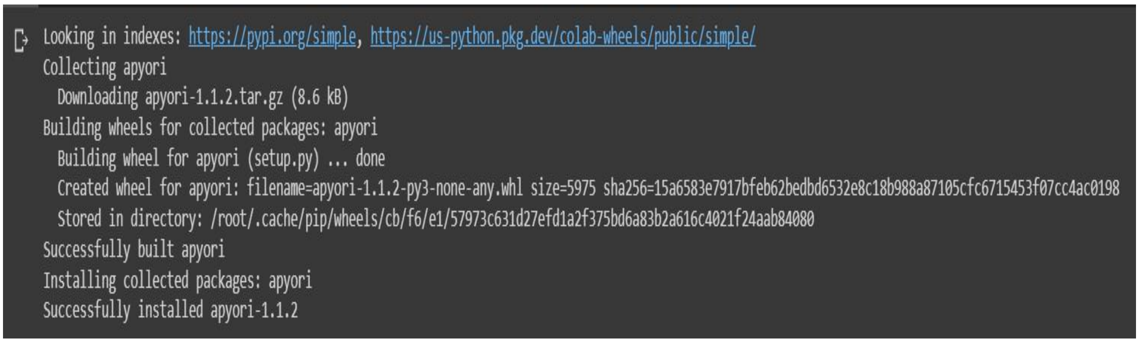
Apriori Property: All non-empty subset of frequent itemset must be frequent. The key concept of Apriori algorithm is its anti-monotonicity of support measure.

Frequent Itemset: The itemset that occurs frequently is called frequent itemset.

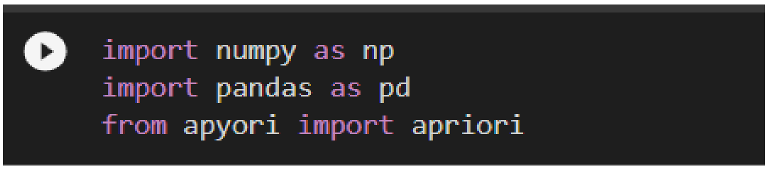
PROGRAM:



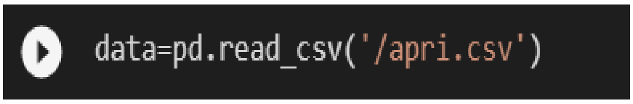
```
pip install apyori
```



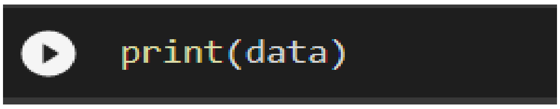
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=5975 sha256=15a6583e7917bfeb62bedbd6532e8c18b988a87105cfc6715453f07cc4ac0198
  Stored in directory: /root/.cache/pip/wheels/cb/f6/e1/57973c631d27efd1a2f375bd6a83b2a616c4021f24aab84080
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```



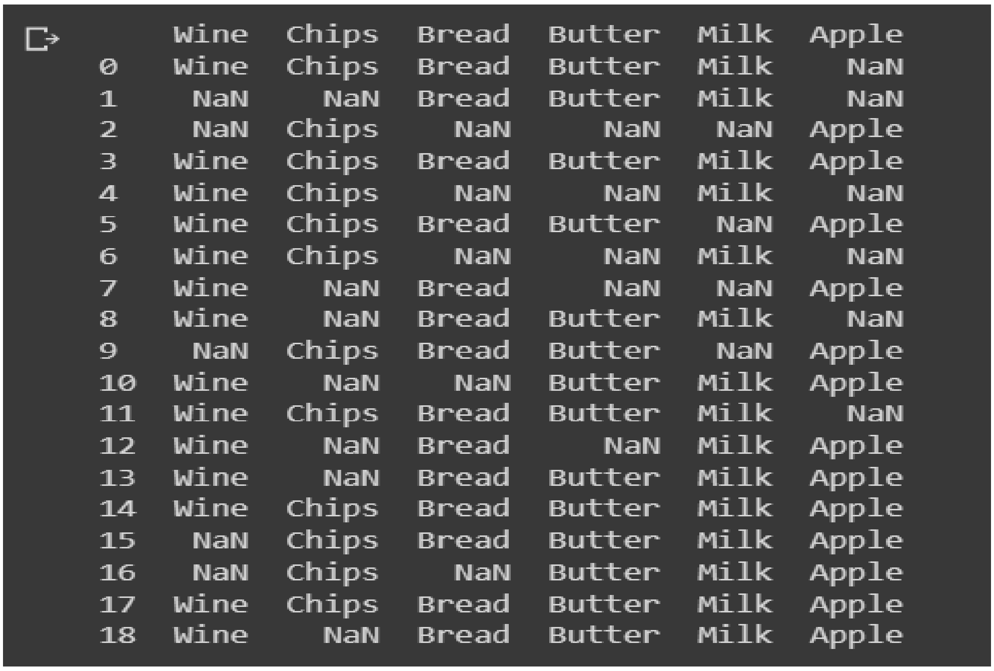
```
import numpy as np
import pandas as pd
from apyori import apriori
```



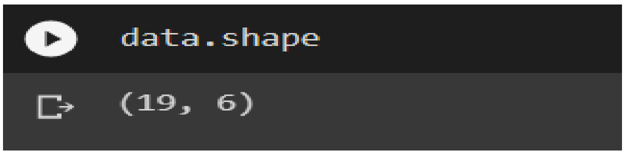
```
data=pd.read_csv('/apri.csv')
```



```
print(data)
```

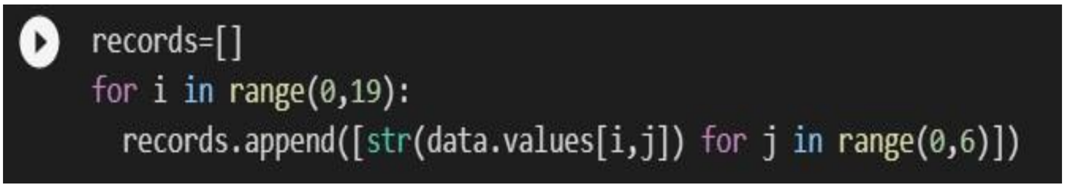


```
      Wine  Chips  Bread  Butter  Milk  Apple
0     Wine  Chips  Bread  Butter  Milk    NaN
1      NaN    NaN  Bread  Butter  Milk    NaN
2      NaN  Chips    NaN     NaN   NaN  Apple
3     Wine  Chips  Bread  Butter  Milk  Apple
4     Wine  Chips    NaN     NaN  Milk    NaN
5     Wine  Chips  Bread  Butter   NaN  Apple
6     Wine  Chips    NaN     NaN  Milk    NaN
7     Wine    NaN  Bread     NaN   NaN  Apple
8     Wine    NaN  Bread  Butter  Milk    NaN
9      NaN  Chips  Bread  Butter   NaN  Apple
10    Wine    NaN    NaN  Butter  Milk  Apple
11    Wine  Chips  Bread  Butter  Milk    NaN
12    Wine    NaN  Bread     NaN  Milk  Apple
13    Wine    NaN  Bread  Butter  Milk  Apple
14    Wine  Chips  Bread  Butter  Milk  Apple
15     NaN  Chips  Bread  Butter  Milk  Apple
16     NaN  Chips    NaN  Butter  Milk  Apple
17    Wine  Chips  Bread  Butter  Milk  Apple
18    Wine    NaN  Bread  Butter  Milk  Apple
```

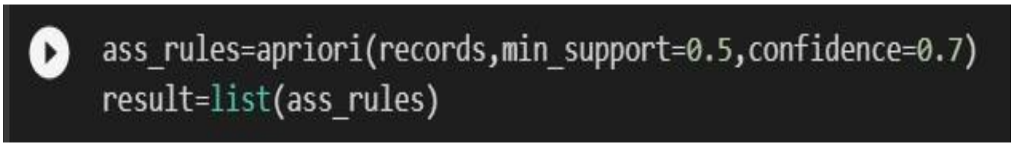


```
data.shape
```

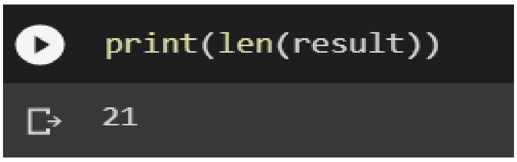
```
(19, 6)
```



```
records=[]
for i in range(0,19):
  records.append([str(data.values[i,j]) for j in range(0,6)])
```

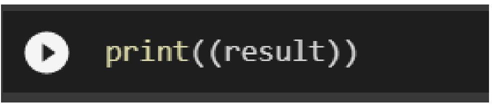


```
ass_rules=apriori(records,min_support=0.5,confidence=0.7)
result=list(ass_rules)
```



```
print(len(result))
```

```
21
```



```
print((result))
```

OUTPUT:

```
[RelationRecord(items=frozenset({'Apple'}), support=0.6842105263157895,
ordered_statistics=[OrderedStatistic(items_base=frozenset(),
items_add=frozenset({'Apple'}), confidence=0.6842105263157895, lift=1.0)]),
RelationRecord(items=frozenset({'Bread'}), support=0.7368421052631579,
ordered_statistics=[OrderedStatistic(items_base=frozenset(),
items_add=frozenset({'Bread'}), confidence=0.7368421052631579, lift=1.0)]),
RelationRecord(items=frozenset({'Butter'}), support=0.7368421052631579,
ordered_statistics=[OrderedStatistic(items_base=frozenset(),
items_add=frozenset({'Butter'}), confidence=0.7368421052631579, lift=1.0)]),
RelationRecord(items=frozenset({'Chips'}), support=0.631578947368421,
ordered_statistics=[OrderedStatistic(items_base=frozenset(),
items_add=frozenset({'Chips'}), confidence=0.631578947368421, lift=1.0)]),
RelationRecord(items=frozenset({'Milk'}), support=0.7894736842105263,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Milk'}),
confidence=0.7894736842105263, lift=1.0)]), RelationRecord(items=frozenset({'Wine'}),
support=0.7368421052631579,
ordered_statistics=[OrderedStatistic(items_base=frozenset(),
items_add=frozenset({'Wine'}), confidence=0.7368421052631579, lift=1.0)]),
RelationRecord(items=frozenset({'nan'}), support=0.8421052631578947,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'nan'}),
confidence=0.8421052631578947, lift=1.0)]), RelationRecord(items=frozenset({'Bread',
'Apple'}), support=0.5263157894736842,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Bread',
'Apple'}), confidence=0.5263157894736842, lift=1.0),
OrderedStatistic(items_base=frozenset({'Apple'}), items_add=frozenset({'Bread'}),
confidence=0.7692307692307692, lift=1.043956043956044),
OrderedStatistic(items_base=frozenset({'Bread'}), items_add=frozenset({'Apple'}),
confidence=0.7142857142857143, lift=1.043956043956044)]),
RelationRecord(items=frozenset({'Butter', 'Apple'}), support=0.5263157894736842,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Butter',
'Apple'}), confidence=0.5263157894736842, lift=1.0),
OrderedStatistic(items_base=frozenset({'Apple'}), items_add=frozenset({'Butter'}),
confidence=0.7692307692307692, lift=1.043956043956044),
OrderedStatistic(items_base=frozenset({'Butter'}), items_add=frozenset({'Apple'}),
confidence=0.7142857142857143, lift=1.043956043956044)]),
RelationRecord(items=frozenset({'nan', 'Apple'}), support=0.5263157894736842,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'nan',
'Apple'}), confidence=0.5263157894736842, lift=1.0),
OrderedStatistic(items_base=frozenset({'Apple'}), items_add=frozenset({'nan'}),
confidence=0.7692307692307692, lift=0.9134615384615384),
OrderedStatistic(items_base=frozenset({'nan'}), items_add=frozenset({'Apple'}),
confidence=0.625, lift=0.9134615384615384)]), RelationRecord(items=frozenset({'Bread',
'Butter'}), support=0.631578947368421,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Bread',
'Butter'}), confidence=0.631578947368421, lift=1.0),
OrderedStatistic(items_base=frozenset({'Bread'}), items_add=frozenset({'Butter'}),
confidence=0.8571428571428571, lift=1.163265306122449),
OrderedStatistic(items_base=frozenset({'Butter'}), items_add=frozenset({'Bread'}),
confidence=0.8571428571428571, lift=1.163265306122449)]),
RelationRecord(items=frozenset({'Bread', 'Milk'}), support=0.5789473684210527,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Bread',
'Milk'}), confidence=0.5789473684210527, lift=1.0),
OrderedStatistic(items_base=frozenset({'Bread'}), items_add=frozenset({'Milk'}),
confidence=0.7857142857142858, lift=0.9952380952380954),
OrderedStatistic(items_base=frozenset({'Milk'}), items_add=frozenset({'Bread'}),
confidence=0.7333333333333334, lift=0.9952380952380954)]),
RelationRecord(items=frozenset({'Bread', 'Wine'}), support=0.5789473684210527,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Bread',
'Wine'}), confidence=0.5789473684210527, lift=1.0),
OrderedStatistic(items_base=frozenset({'Bread'}), items_add=frozenset({'Wine'}),
confidence=0.7857142857142858, lift=1.0663265306122451),
OrderedStatistic(items_base=frozenset({'Wine'}), items_add=frozenset({'Bread'}),
confidence=0.7857142857142858, lift=1.0663265306122451)]),
RelationRecord(items=frozenset({'Bread', 'nan'}), support=0.5789473684210527,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Bread',
'nan'}), confidence=0.5789473684210527, lift=1.0),
OrderedStatistic(items_base=frozenset({'Bread'}), items_add=frozenset({'nan'}),
confidence=0.7857142857142858, lift=0.9330357142857144),
OrderedStatistic(items_base=frozenset({'nan'}), items_add=frozenset({'Bread'}),
confidence=0.6875000000000001, lift=0.9330357142857145)]),
RelationRecord(items=frozenset({'Butter', 'Milk'}), support=0.631578947368421,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Butter',
'Milk'}), confidence=0.631578947368421, lift=1.0),
OrderedStatistic(items_base=frozenset({'Butter'}), items_add=frozenset({'Milk'}),
confidence=0.8571428571428571, lift=1.0857142857142856),
OrderedStatistic(items_base=frozenset({'Milk'}), items_add=frozenset({'Butter'}),
confidence=0.7999999999999999, lift=1.0857142857142856)]),
RelationRecord(items=frozenset({'Butter', 'Wine'}), support=0.5263157894736842,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Butter',
'Wine'}), confidence=0.5263157894736842, lift=1.0),
OrderedStatistic(items_base=frozenset({'Butter'}), items_add=frozenset({'Wine'}),
confidence=0.7142857142857143, lift=0.9693877551020409),
OrderedStatistic(items_base=frozenset({'Wine'}), items_add=frozenset({'Butter'}),
confidence=0.7142857142857143, lift=0.9693877551020409)]),
RelationRecord(items=frozenset({'nan', 'Butter'}), support=0.5789473684210527,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'nan',
'Butter'}), confidence=0.5789473684210527, lift=1.0),
OrderedStatistic(items_base=frozenset({'Butter'}), items_add=frozenset({'nan'}),
confidence=0.7857142857142858, lift=0.9330357142857144),
OrderedStatistic(items_base=frozenset({'nan'}), items_add=frozenset({'Butter'}),
confidence=0.6875000000000001, lift=0.9330357142857145)]),
RelationRecord(items=frozenset({'Milk', 'Wine'}), support=0.631578947368421,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Milk',
'Wine'}), confidence=0.631578947368421, lift=1.0),
OrderedStatistic(items_base=frozenset({'Milk'}), items_add=frozenset({'Wine'}),
confidence=0.7999999999999999, lift=1.0857142857142856),
OrderedStatistic(items_base=frozenset({'Wine'}), items_add=frozenset({'Milk'}),
confidence=0.8571428571428571, lift=1.0857142857142856)]),
RelationRecord(items=frozenset({'nan', 'Milk'}), support=0.631578947368421,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'nan',
'Milk'}), confidence=0.631578947368421, lift=1.0),
OrderedStatistic(items_base=frozenset({'Milk'}), items_add=frozenset({'nan'}),
confidence=0.7999999999999999, lift=0.95),
OrderedStatistic(items_base=frozenset({'nan'}), items_add=frozenset({'Milk'}),
confidence=0.75, lift=0.95)]), RelationRecord(items=frozenset({'nan', 'Wine'}),
support=0.5789473684210527,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'nan',
'Wine'}), confidence=0.5789473684210527, lift=1.0),
OrderedStatistic(items_base=frozenset({'Wine'}), items_add=frozenset({'nan'}),
confidence=0.7857142857142858, lift=0.9330357142857144),
OrderedStatistic(items_base=frozenset({'nan'}), items_add=frozenset({'Wine'}),
confidence=0.6875000000000001, lift=0.9330357142857145)]),
RelationRecord(items=frozenset({'Bread', 'Butter', 'Milk'}), support=0.5263157894736842,
ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Bread',
'Butter', 'Milk'}), confidence=0.5263157894736842, lift=1.0),
OrderedStatistic(items_base=frozenset({'Bread'}), items_add=frozenset({'Butter', 'Milk'}),
confidence=0.7142857142857143, lift=1.130952380952381),
OrderedStatistic(items_base=frozenset({'Butter'}), items_add=frozenset({'Bread', 'Milk'}),
confidence=0.7142857142857143, lift=1.2337662337662338),
OrderedStatistic(items_base=frozenset({'Milk'}), items_add=frozenset({'Bread', 'Butter'}),
confidence=0.6666666666666666, lift=1.0555555555555556),
OrderedStatistic(items_base=frozenset({'Bread', 'Butter'}), items_add=frozenset({'Milk'}),
confidence=0.8333333333333334, lift=1.0555555555555556),
OrderedStatistic(items_base=frozenset({'Bread', 'Milk'}), items_add=frozenset({'Butter'}),
confidence=0.909090909090909, lift=1.2337662337662336),
OrderedStatistic(items_base=frozenset({'Butter', 'Milk'}), items_add=frozenset({'Bread'}),
confidence=0.8333333333333334, lift=1.1309523809523812)])]
```

Week – 11

Write a Python program to generate frequent item sets / association rules using FP-growth Tree algorithm.

AIM: To write a Python program to generate frequent item sets / association rules using FP-growth Tree algorithm.

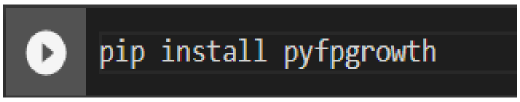
DECRIPTION:

Frequent Itemset: The itemset that occurs frequently is called frequent itemset.

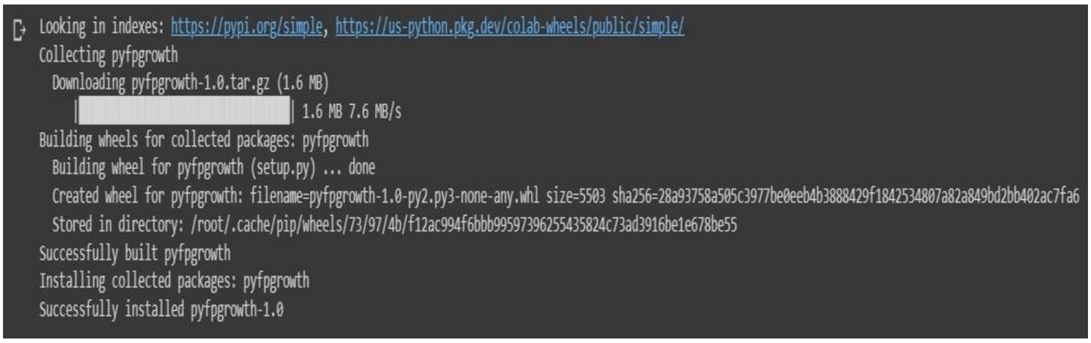
The FP-Growth Algorithm is an alternative way to find frequent item sets without using candidate generations. For so much, it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information.

FP-Tree: The frequent-pattern tree (FP-tree) is a compact data structure that stores quantitative information about frequent patterns in a database. Each transaction is read and then mapped onto a path in the FP-tree. This is done until all transactions have been read.

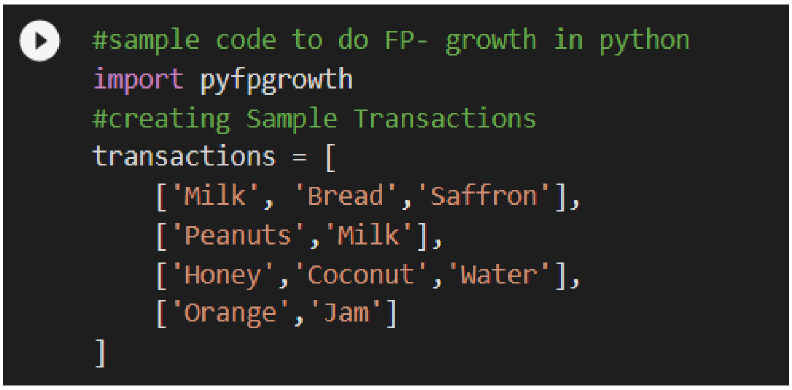
PROGRAM:



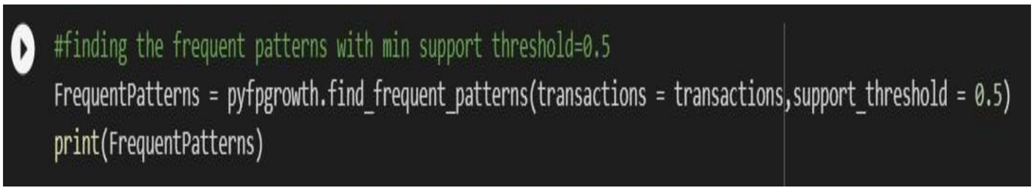
```
pip install pyfpgrowth
```



```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyfpgrowth
  Downloading pyfpgrowth-1.0.tar.gz (1.6 MB)
     |████████████████████████████████| 1.6 MB 7.6 MB/s
Building wheels for collected packages: pyfpgrowth
  Building wheel for pyfpgrowth (setup.py) ... done
  Created wheel for pyfpgrowth: filename=pyfpgrowth-1.0-py2.py3-none-any.whl size=5503 sha256=28a93758a505c3977be0eeb4b3888429f1842534807a82a849bd2bb402ac7fa6
  Stored in directory: /root/.cache/pip/wheels/73/97/4b/f12ac994f6bbb99597396255435824c73ad3916be1e678be55
Successfully built pyfpgrowth
Installing collected packages: pyfpgrowth
Successfully installed pyfpgrowth-1.0
```



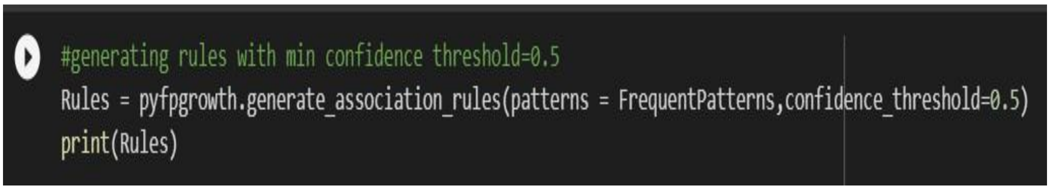
```
#sample code to do FP- growth in python
import pyfpgrowth
#creating Sample Transactions
transactions = [
    ['Milk', 'Bread','Saffron'],
    ['Peanuts','Milk'],
    ['Honey','Coconut','Water'],
    ['Orange','Jam']
]
```



```
#finding the frequent patterns with min support threshold=0.5
FrequentPatterns = pyfpgrowth.find_frequent_patterns(transactions = transactions,support_threshold = 0.5)
print(FrequentPatterns)
```

OUTPUT:

```
{('Bread',): 1, ('Bread', 'Milk'): 1, ('Saffron',): 1, ('Bread', 'Saffron'): 1, ('Milk', 'Saffron'): 1,
('Bread', 'Milk', 'Saffron'): 1, ('Peanuts',): 1, ('Milk', 'Peanuts'): 1, ('Honey',): 1, ('Coconut',): 1,
('Coconut', 'Honey'): 1, ('Water',): 1, ('Coconut', 'Water'): 1, ('Honey', 'Water'): 1, ('Coconut',
'Honey', 'Water'): 1, ('Orange',): 1, ('Jam',): 1, ('Jam', 'Orange'): 1, ('Milk',): 2}
```



```
#generating rules with min confidence threshold=0.5
Rules = pyfpgrowth.generate_association_rules(patterns = FrequentPatterns,confidence_threshold=0.5)
print(Rules)
```

OUTPUT:

```
{('Bread',): (('Milk', 'Saffron'), 1.0), ('Milk',): (('Peanuts',), 0.5), ('Saffron',): (('Bread', 'Milk'),
1.0), ('Bread', 'Milk'): (('Saffron',), 1.0), ('Bread', 'Saffron'): (('Milk',), 1.0), ('Milk', 'Saffron'):
(('Bread',), 1.0), ('Peanuts',): (('Milk',), 1.0), ('Coconut',): (('Honey', 'Water'), 1.0), ('Honey',):
(('Coconut', 'Water'), 1.0), ('Water',): (('Coconut', 'Honey'), 1.0), ('Coconut', 'Honey'):
(('Water',), 1.0), ('Coconut', 'Water'): (('Honey',), 1.0), ('Honey', 'Water'): (('Coconut',), 1.0),
('Jam',): (('Orange',), 1.0), ('Orange',): (('Jam',), 1.0)}
```