Shawn Cicoria - CS 634 Data Mining Midterm Project

Shawn Cicoria

March 12 2020

# Py\_Apriori Module

This is a basic implementation in Python of the Apriori Association Algorithm

## Requirements

* Linux / Ubuntu 16+
* Python 3.6

## Setup

First download the TAR file to a location that you will use. I recommend a scratch directory that you can remove all when done.

In the text below, my extracted directory is: /c/temp/py-apriori-0.1.0

### Setup your Python environment

General recommendation is to use a Python Virtual Environment. With Python 3.6+ execute the following:

#### Create virtual env and activate

# from /c/temp/  
python -m venv env # this creates directory env  
. ./env/bin/activate

### Unpack the tar file

From the directory where the virtual env and the tar file is:

# from /c/temp/  
tar -xvf py-apriori-0.1.0.tar.gz  
cd py-apriori-0.1.0.tar.gz

### Run setup

Once you’ve extracted the tar file, and changed to the tar output directory, now run setup.py install under python to add all dependencies

# from /c/temp/py-apriori-0.1.0  
  
python setup.py install

At this point the program is ready to run, and a Test data file is present in ./data/

## Running

The program makes use of argument parsing and all arguments can be seen by running the following:

# from /c/temp/py-apriori-0.1.0  
  
python main.py -h  
  
# ---  
usage: main.py [-h] -i FILE [-c CONFIDENCE\_LEVEL] [-s SUPPORT\_LEVEL] [-n]  
 [-o FILE]  
  
implementation of the Apriori Algorithm  
  
optional arguments:  
 -h, --help show this help message and exit  
 -i FILE, --input FILE  
 input transaction file collapsed CSV format  
 -c CONFIDENCE\_LEVEL, --confidence CONFIDENCE\_LEVEL  
 confidence level for association generation see https://en.wikipedia.org/wiki/Association\_rule\_learning#Confidence  
 -s SUPPORT\_LEVEL, --support SUPPORT\_LEVEL  
 support level for support generation see https://en.wikipedia.org/wiki/Association\_rule\_learning#Support  
 -n, --no-drop DO NOT drop transactions below support level  
 -o FILE, --output FILE  
 output file

### Options

|  |  |  |  |
| --- | --- | --- | --- |
| option | description | required | default |
| -i or –input | specifies the input file that MUST be in CollapsedCSV format - see file format section below. | YES | na |
| -c or –confidence\_level | sets the filtering criteria for associations that fall below the specified level. | NO | 0.80 |
| -s or –support\_level | sets the filtering criteria for support levels in the transactions that fall below the specified level. | NO | 0.20 |
| -n or –no-drop | indicate IF you want to included items that fall below the support level in support generation | NO | - if this flag is present then ALL transactions at ALL support levels filter through to the confidence and association generation |

### 

### Sample run

The extracted TAR file has a sample input file in ./data – to run:

# from /c/temp/py-apriori-0.1.0  
python main.py -i data/data.csv

#### Sample Run Output

cicorias@cicoria-msi:/c/temp/py-apriori-0.1.0$ python main.py -i data/data.csv  
For this run we are using the following  
  
 Support: 0.2  
 Confidence: 0.8  
 Drop Trans: True  
 File: /c/temp/py-apriori-0.1.0/data/data.csv  
  
  
=== SUPPORT LEVELS ===  
  
 itemsets count support  
0 (I1,) 6 0.666667  
1 (I2,) 7 0.777778  
2 (I3,) 6 0.666667  
3 (I4,) 2 0.222222  
4 (I5,) 2 0.222222  
5 (I1, I2) 4 0.444444  
6 (I1, I3) 4 0.444444  
7 (I1, I5) 2 0.222222  
8 (I2, I3) 4 0.444444  
9 (I2, I4) 2 0.222222  
10 (I2, I5) 2 0.222222  
11 (I1, I2, I3) 2 0.222222  
12 (I1, I2, I5) 2 0.222222  
  
  
=== ASSOCIATION AND CONFIDENCE LEVELS ===  
  
 full\_key predecessor support1 result support2 support\_full\_key confidence  
23 (I4, I2) (I4,) 0.222222 (I2,) 0.777778 0.222222 1.0  
33 (I5, I1) (I5,) 0.222222 (I1,) 0.666667 0.222222 1.0  
34 (I5, I2) (I5,) 0.222222 (I2,) 0.777778 0.222222 1.0  
37 (I5, I1, I2) (I5,) 0.222222 (I1, I2) 0.444444 0.222222 1.0  
50 (I1, I5, I2) (I1, I5) 0.222222 (I2,) 0.777778 0.222222 1.0  
64 (I2, I5, I1) (I2, I5) 0.222222 (I1,) 0.666667 0.222222 1.0

### Sample Run Data File

I1, I2, I5  
I2, I4  
I2, I3  
I1, I2, I4  
I1, I3  
I2, I3  
I1, I3  
I1, I2, I3, I5  
I1, I2, I3

## Data Format

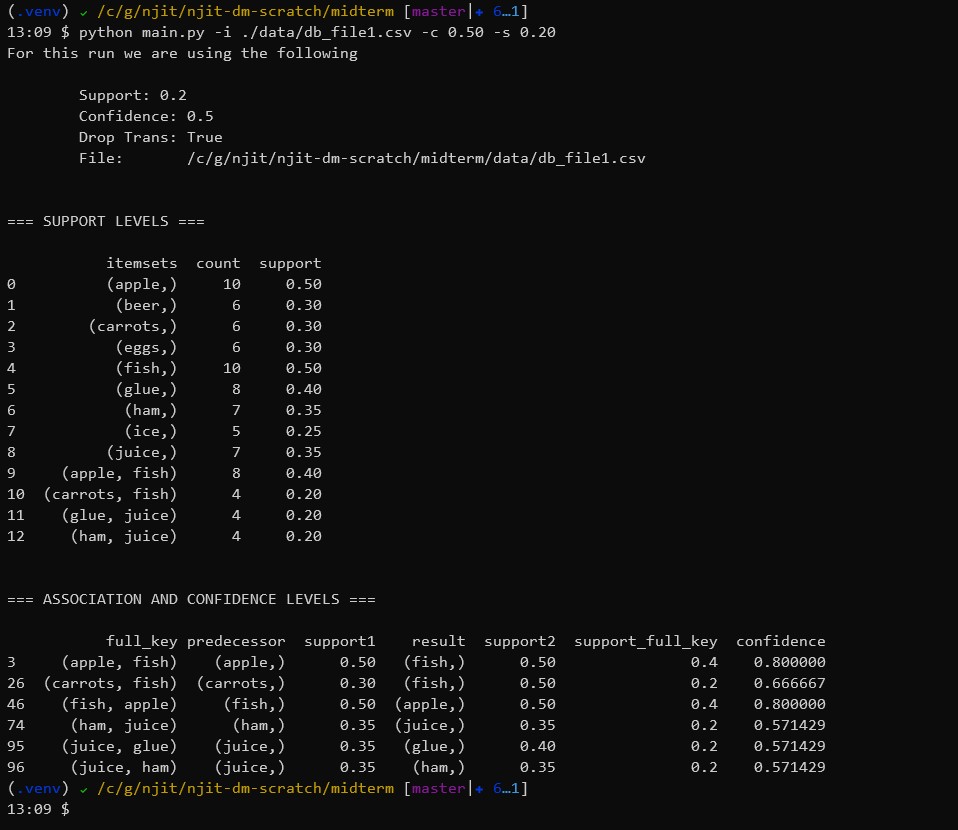
The data file is in a simple format that I call **Collapsed CSV** as each line has multiple transaction items separated by a comma. So, it’s not exactly a CSV file, but close.

# Project Output

The following are the test DB and the generated console output from the program. ## Test DB 1 ### DB File

diapers,eggs,fish,  
apple,fish,glue,ham,juice,  
apple,beer,carrots,diapers,fish,  
eggs,ham,ice,  
carrots,glue,ham,  
apple,fish,juice,  
glue,ham,juice,  
beer,eggs,ice,  
carrots,fish,ice,  
beer,glue,juice,  
apple,fish,glue,  
beer,ham,juice,  
apple,carrots,fish,ham,juice,  
apple,fish,ice,  
apple,eggs,glue,  
apple,diapers,eggs,  
beer,carrots,glue,  
apple,carrots,eggs,fish,ham,  
apple,beer,fish,  
glue,ice,juice,

### Output

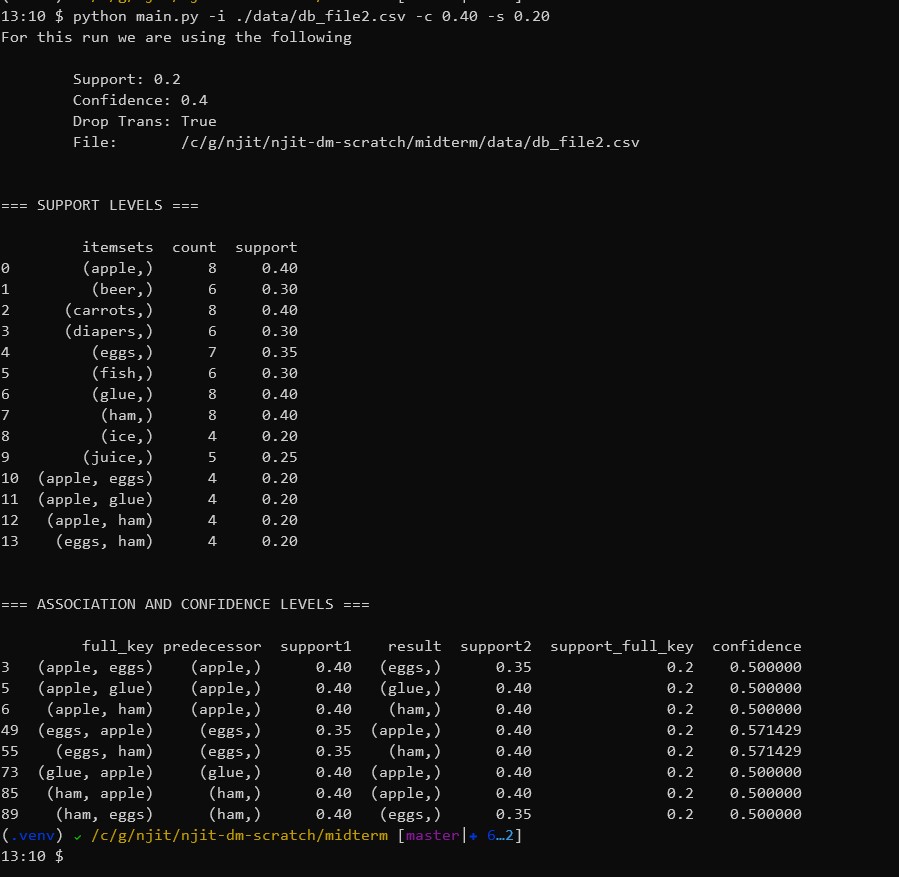


## Test DB 2

### DB File

beer,carrots,glue,  
beer,ham,juice,  
beer,carrots,glue,  
diapers,fish,juice,  
carrots,eggs,ham,  
carrots,fish,glue,  
diapers,fish,ice,  
diapers,ham,juice,  
beer,carrots,ice,  
apple,diapers,eggs,glue,ice,  
apple,beer,eggs,glue,ham,  
apple,beer,ham,  
carrots,eggs,ham,  
carrots,eggs,fish,  
apple,glue,juice,  
apple,carrots,diapers,  
apple,eggs,glue,  
fish,glue,juice,  
apple,diapers,eggs,ham,ice,  
apple,fish,ham,

### Output

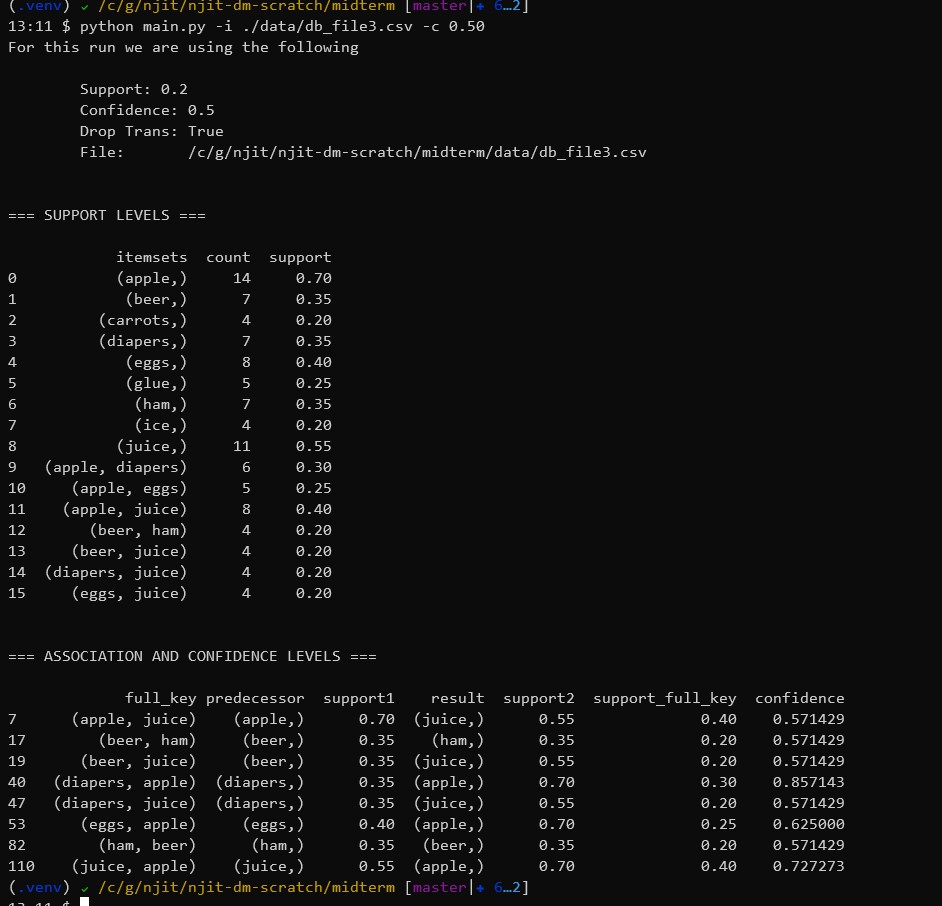


## Test DB 3

### DB File

beer,ham,juice,  
apple,diapers,eggs,ice,  
apple,diapers,juice,  
beer,carrots,ham,  
apple,beer,juice,  
apple,carrots,diapers,juice,  
beer,eggs,glue,ham,  
apple,carrots,ice,juice,  
apple,diapers,eggs,glue,  
apple,diapers,juice,  
apple,glue,ham,  
apple,beer,eggs,juice,  
beer,glue,ham,  
apple,eggs,juice,  
apple,diapers,glue,  
apple,carrots,eggs,ham,  
diapers,eggs,juice,  
apple,ham,ice,  
eggs,ice,juice,  
apple,beer,juice,

### Output

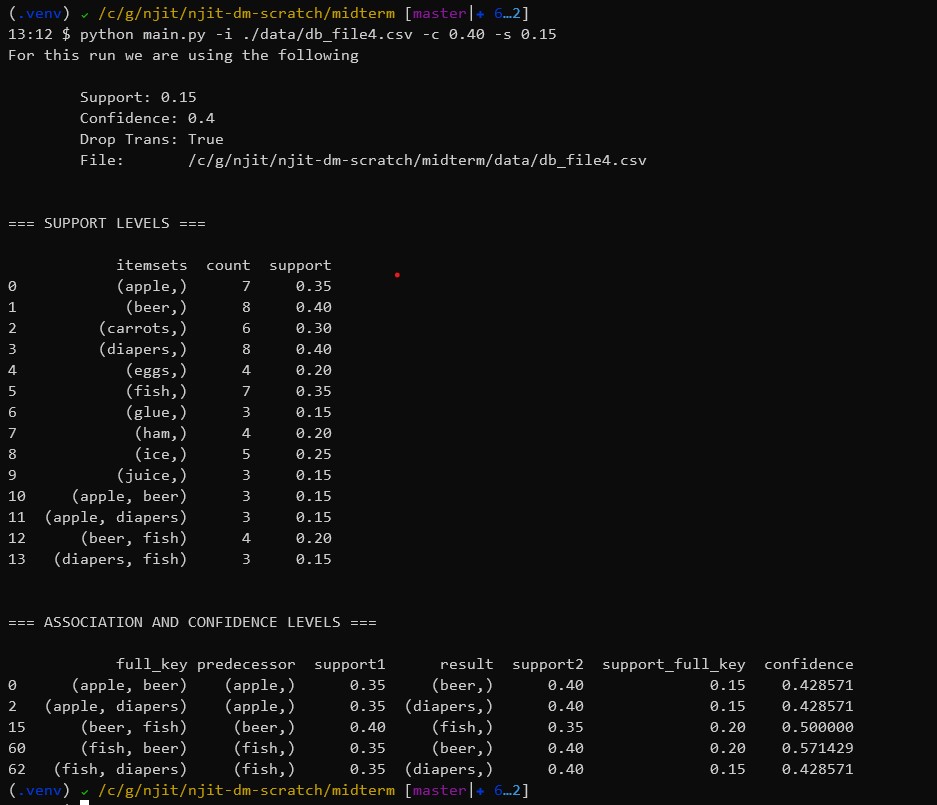


## Test DB 4

### DB File

eggs,glue,  
apple,eggs,ham,  
apple,glue,  
beer,fish,  
carrots,diapers,  
apple,diapers,ice,  
carrots,diapers,eggs,  
apple,diapers,fish,  
diapers,ice,  
carrots,ice,  
beer,diapers,fish,  
apple,beer,diapers,fish,juice,  
beer,eggs,juice,  
apple,beer,ice,  
carrots,fish,juice,  
beer,carrots,ham,  
apple,beer,carrots,  
fish,ice,  
beer,fish,ham,  
diapers,glue,ham,

### Output

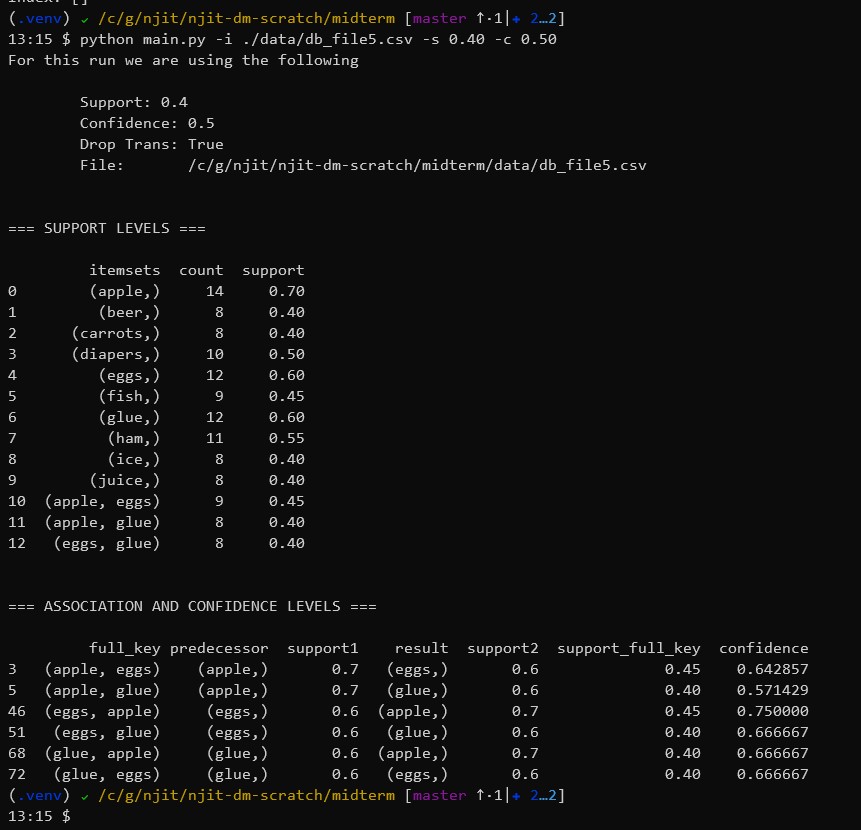


## Test DB 5

### DB File

apple,carrots,eggs,glue,juice,  
apple,beer,diapers,ice,juice,  
beer,carrots,diapers,glue,ham,  
apple,carrots,fish,glue,ice,  
apple,beer,ham,ice,juice,  
apple,eggs,fish,glue,ham,  
apple,eggs,fish,glue,juice,  
apple,diapers,fish,ham,ice,  
apple,diapers,eggs,glue,ham,  
apple,carrots,eggs,fish,ice,  
carrots,diapers,eggs,ham,ice,  
beer,carrots,eggs,fish,glue,  
apple,diapers,eggs,fish,ham,  
beer,diapers,eggs,glue,juice,  
apple,diapers,eggs,glue,ice,  
apple,beer,carrots,glue,ham,  
apple,diapers,eggs,glue,ice,  
beer,diapers,glue,ham,juice,  
beer,carrots,fish,ham,juice,  
apple,eggs,fish,ham,juice,

### Output



# Program Files

## main.py

import os  
import sys  
import argparse  
import logging  
  
from py\_apriori.apriori import Apriori, CollapsedCsvFileReader  
from py\_apriori.assoc import (calculate\_confidence,  
 create\_associations)  
  
logging.basicConfig()  
logger = logging.getLogger('apriori')  
logger.setLevel(logging.WARN)  
  
  
class Program:  
 def \_\_init\_\_(self):  
 self.data = []  
  
 def parse\_arguments(self):  
 parser = argparse.ArgumentParser(description='implementation of the Apriori Algorithm',  
 formatter\_class=argparse.RawTextHelpFormatter)  
 parser.add\_argument('-i', '--input', dest='FILE', required=True,  
 help='input transaction file collapsed CSV format', metavar='FILE',  
 type=lambda x: self.is\_valid\_file(parser, x))  
 parser.add\_argument('-c', '--confidence', dest='confidence\_level', required=False,  
 default=0.80,  
 type=float,  
 help='confidence level for association generation see https://en.wikipedia.org/wiki/Association\_rule\_learning#Confidence')  
 parser.add\_argument('-s', '--support', dest='support\_level', required=False,  
 default=0.20,  
 type=float,  
 help='support level for support generation see https://en.wikipedia.org/wiki/Association\_rule\_learning#Support')  
 parser.add\_argument('-n', '--no-drop', dest='drop\_below\_support\_level', required=False,  
 default=True, action='store\_false',  
 help='DO NOT drop transactions below support level')  
  
 parser.add\_argument("-o", "--output", dest="output",  
 type=argparse.FileType('w'),  
 metavar="FILE",  
 default=sys.stdout,  
 help="output file")  
  
 self.args = parser.parse\_args()  
  
 def is\_valid\_file(self, parser, arg):  
 if not os.path.exists(arg):  
 parser.error("The file %s does not exist!" % arg)  
 else:  
 return os.path.abspath(arg)  
  
 @property  
 def FILE(self):  
 return self.args.FILE  
  
 def print(self, content):  
 print(content, file=self.args.output)  
  
  
def main():  
 prog = Program()  
 prog.parse\_arguments()  
  
 file\_reader = CollapsedCsvFileReader(prog.FILE)  
 raw\_transactions = file\_reader.read()  
  
 apriori\_instance = Apriori(raw\_transactions)  
  
 # setup for output  
 support = prog.args.support\_level  
 confidence = prog.args.confidence\_level  
 drop\_trans = prog.args.drop\_below\_support\_level  
  
 prog.print("For this run we are using the following\n")  
 prog.print("\tSupport: {}".format(support))  
 prog.print("\tConfidence: {}".format(confidence))  
 prog.print("\tDrop Trans: {}".format(drop\_trans))  
 prog.print("\tFile: {}".format(prog.FILE))  
  
 # just generate the levels and filter as needed  
 support\_level\_output = apriori\_instance.generate\_levels(support\_level=support, drop\_below\_support=drop\_trans)  
 prog.print("\n\n=== SUPPORT LEVELS ===\n")  
 prog.print(support\_level\_output)  
  
 # create the associations  
 # TODO: mabye encapsulate this step.  
 associated\_transactions = create\_associations(support\_level\_output)  
  
 # generate the confidence levels.  
 confidence\_report = calculate\_confidence(associated\_transactions, confidence\_level=confidence)  
  
 prog.print("\n\n=== ASSOCIATION AND CONFIDENCE LEVELS ===\n")  
 prog.print(confidence\_report)  
  
 # prog.print(len(confidence\_report))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## py\_apriori/apriori.py

from typing import List, Tuple  
from itertools import combinations  
from abc import ABC, abstractmethod  
import pandas as pd  
import logging  
  
logging.basicConfig()  
logger = logging.getLogger("apriori")  
  
  
class Apriori(object):  
 # following https://en.wikipedia.org/wiki/Apriori\_algorithm#Examples  
 def \_\_init\_\_(self, transactions: List):  
 self.\_\_verify\_\_(transactions)  
  
 self.\_transactions = transactions  
  
 def \_\_str\_\_(self):  
 return str(tuple(self))  
  
 def \_\_verify\_\_(self, transactions):  
 if transactions is None:  
 raise ValueError("Transaction itemset is none")  
  
 if not isinstance(transactions, List):  
 raise ValueError("Transaction itemset is not a List")  
  
 if len(transactions) == 0:  
 raise ValueError("Transaction is empty")  
  
 if len(transactions) > 0 and not isinstance(transactions[0], Tuple):  
 raise ValueError("Transaction lement is not a Tuple")  
  
 @property  
 def transactions(self) -> List:  
 return self.\_transactions  
  
 @transactions.setter  
 def transactions(self, value: List):  
  
 self.\_\_verify\_\_(value)  
 self.\_transactions = value  
  
 def generate\_levels(self, support\_level: float = 0.20, drop\_below\_support: bool = True) -> pd.DataFrame:  
 k = 1  
 full\_set = list() # this contains a dataframe for each level.  
 while True:  
 logger.info("k = {0}".format(k))  
 item\_levels = self.\_\_generate\_combinination\_levels(self.\_transactions, k)  
 sl = self.\_\_gen\_support\_level(self.\_transactions, item\_levels,  
 support=support\_level, drop=drop\_below\_support)  
  
 # logger.debug("transactions at level {k} are {n}".format(k = k, n = (len(sl))))  
 k += 1  
  
 if len(sl) == 0 or k == 100:  
 break  
  
 df = pd.DataFrame.from\_dict(sl, orient='index', columns=['count'])  
 df.index.name = 'itemsets'  
 df.reset\_index()  
 full\_set.append(df)  
  
 rv = self.\_\_append\_colums(full\_set)  
 return rv  
  
 def \_\_append\_colums(self, data: List, tran\_list=None) -> pd.DataFrame:  
 if tran\_list is None:  
 tran\_list = self.transactions  
  
 tran\_count = len(tran\_list)  
  
 rows\_list = []  
 for r in data:  
 # logger.debug('type of r is: {0}'.format(type(r)))  
 # logger.debug('len of r is: {0}'.format(len(r)))  
 # logger.debug('r is: {0}'.format(r))  
 for index, row in r.iterrows():  
 # d = { 'count' : r['count'], 'support': r['count']/tran\_count}  
 d = {'itemsets': index, 'count': row['count'], 'support': row['count']/tran\_count}  
 # logger.debug("THE DICTd: {0}".format(d))  
 rows\_list.append(d)  
  
 df = pd.DataFrame(rows\_list)  
  
 return df  
  
 def \_\_generate\_combinination\_levels(self, tran\_list, level):  
 """generate keys that are used for subset checking"""  
 """on each transaction"""  
 results = list()  
 for t in tran\_list:  
 logger.debug("gen\_com\_levell: t: {0} and level: {1}".format(t, level))  
 [results.append(i) for i in combinations(t, level)]  
  
 rv = sorted(set(results))  
 logger.debug("combo levels: {0}".format(rv))  
 return rv  
  
 def \_\_gen\_support\_level(self, tran\_list, items\_keys, support=0.20, drop=True):  
 """for each key which can be a set find in transactions"""  
 """how many contain the combination"""  
 logger.info('Using support level of {0}'.format(support))  
 logger.info('drop below support? {0}'.format(drop))  
 tran\_count = len(tran\_list)  
 base\_level = tran\_count \* support  
 logger.debug('base level count: {0}'.format(base\_level))  
 itemSet = dict()  
  
 for key in items\_keys:  
 for t in tran\_list:  
 if set(key).issubset(t):  
 # logger.debug('is subset: {0}'.format(t))  
 if (key) in itemSet:  
 itemSet[key] += 1  
 else:  
 itemSet[key] = 1  
  
 if drop:  
 return {key: value for (key, value) in itemSet.items() if value >= base\_level}  
 else:  
 return {key: value for (key, value) in itemSet.items()}  
  
  
class FileReader(ABC):  
 def \_\_init\_\_(self, file\_path):  
 self.file\_path = file\_path  
  
 @abstractmethod  
 def read(self) -> list:  
 pass  
  
  
class CollapsedCsvFileReader(FileReader):  
 """the file format is lines, with individual transactinos"""  
 """separated by commma - thus calling this collapsed"""  
 """file format as it is non-traditional"""  
  
 def read(self) -> list:  
 file\_iter = open(self.file\_path, 'r')  
 raw\_transactions = list()  
 for line in file\_iter:  
 line = line.strip().rstrip(',')  
 # remove whitespace around items  
 trimmed = [i.strip() for i in line.split(',')]  
 record = tuple(sorted(trimmed))  
 raw\_transactions.append(record)  
  
 return raw\_transactions

## py\_apriori/assoc.py

from typing import Tuple  
from collections import namedtuple  
  
import pandas as pd  
import logging  
  
logging.basicConfig()  
logger = logging.getLogger("apriori")  
  
  
assocation\_record = namedtuple('assocation\_record', ['full\_key', 'predecessor',  
 'support1', 'result', 'support2', 'support\_full\_key', 'confidence'])  
  
  
def create\_associations(data: pd.DataFrame) -> pd.DataFrame:  
 n2 = generate\_associations(data)  
 pc = generate\_combo\_itemsets(n2)  
 return pc  
  
  
def generate\_associations(data: pd.DataFrame) -> pd.DataFrame:  
 rv = list()  
 # TODO: refactor itertuples  
 for r in data.iterrows():  
 # current row ID  
 idx = r[0]  
 item = data.iloc[idx]['itemsets']  
 # all\_other is everything BUT the current key.  
 all\_other = [k for k, v in data.iterrows() if k != idx]  
 # THIS current item support.  
 support = data.iloc[idx]['support']  
  
 temp = [(item, support, data.iloc[y]['itemsets'], data.iloc[y]['support'])  
 for y in all\_other  
 if not set(item).issubset(data.iloc[y]['itemsets'])]  
  
 rv.extend(temp)  
  
 return pd.DataFrame(rv, columns=['predecessor', 'support1', 'result', 'support2'])  
  
  
def generate\_combo\_itemsets(data: pd.DataFrame) -> pd.DataFrame:  
 possible\_combos = [(i[1]['predecessor'] + i[1]['result'],  
 i[1]['predecessor'], i[1]['support1'], i[1]['result'], i[1]['support2'])  
 # TODO: refactor itertuples  
 for i in data.iterrows()  
 # elimiate where item on boths sides  
 if not bool(set(i[1]['result']) & set(i[1]['predecessor']))]  
  
 return pd.DataFrame(possible\_combos, columns=['fullkey', 'predecessor', 'support1', 'result', 'support2'])  
  
  
def get\_support\_for\_key(data: pd.DataFrame, predecessor\_key: Tuple) -> float:  
 rv = 0  
 srted\_key = tuple(sorted(predecessor\_key))  
 matches = data[data['predecessor'] == srted\_key].head(1)  
 if len(matches) == 1:  
 rv = matches['support1'].array[0]  
  
 return rv  
  
  
def calculate\_confidence(data: pd.DataFrame, confidence\_level: float = 0.0) -> pd.DataFrame:  
 rv = list()  
 for r in data.itertuples():  
 full\_key = r.fullkey  
 ant = r.predecessor  
 support1 = r.support1  
 res = r.result  
 support2 = r.support2  
 support\_full\_key = get\_support\_for\_key(data, full\_key)  
 if support1 != 0:  
 confidence = support\_full\_key / support1  
 else:  
 confidence = -1  
  
 item\_rv = assocation\_record(  
 full\_key, ant, support1, res, support2, support\_full\_key, confidence)  
  
 rv.append(item\_rv)  
  
 rv\_df = pd.DataFrame(rv)  
 return rv\_df[rv\_df.confidence > confidence\_level]

## requirements.txt

poetry==1.0.2  
pandas==1.0.1

## setup.cfg

[flake8]  
max-line-length = 160

## Tools

### gen.py

from itertools import combinations  
import random  
  
  
def read(file\_path) -> list:  
 file\_iter = open(file\_path, 'r')  
 items = list()  
 for line in file\_iter:  
 line = line.strip()  
 items.append(line)  
  
 return items  
  
  
def get\_random\_count():  
 return random.randint(2, 5)  
  
  
def generate\_itemset(input\_file):  
 rv = list()  
 out\_list = list()  
 for r in range(0, 20):  
 item\_permutations = [out\_list.append(i) for i in combinations(input\_file, get\_random\_count())]  
 total\_permutations = len(item\_permutations)  
 ic = random.randint(0, total\_permutations)  
 item\_set = out\_list[ic]  
 rv.append(sorted(item\_set))  
  
 return rv  
  
  
def generate\_db\_file(input\_file, output\_file):  
 file1 = generate\_itemset(input\_file)  
 with open(output\_file, "w") as outfile:  
 all\_buffer = ""  
 for item in file1:  
 buffer = ""  
 for i in item:  
 buffer += i + ","  
  
 buffer.strip().rstrip(',')  
 all\_buffer += buffer + '\n'  
  
 outfile.writelines(all\_buffer)  
  
  
random.seed(2020)  
input\_file = read('./data/item.csv')  
  
generate\_db\_file(input\_file, './data/db\_file1.csv')  
generate\_db\_file(input\_file, './data/db\_file2.csv')  
generate\_db\_file(input\_file, './data/db\_file3.csv')  
generate\_db\_file(input\_file, './data/db\_file4.csv')  
generate\_db\_file(input\_file, './data/db\_file5.csv')

# References:

This is a basic implementation of the Apriori Algorithm[1]

[Google Scholar](https://scholar.google.com/scholar?q=R.C.%20Agarwal%2C%20C.C.%20Aggarwal%2C%20and%20V.V.V.%20Prasad.%20Depth%20first%20generation%20of%20long%20patterns.%20In%20Proc.%20of%20the%206th%20ACM%20SIGKDD%20Int.%20Conf.%20on%20Knowledge%20Discovery%20and%20Data%20Mining%2C%20pages%20108%E2%80%93118%2C%20Boston%2C%20MA%2C%20USA%2C%202000.) - Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami. “Mining association rules between sets of items in large databases.” Proceedings of the 1993 ACM SIGMOD international conference on Management of data. 1993.