In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

Question 1

In [13]:

```
data = pd.read_csv('car.data',names=['0','1','2','3','4','5','6'])
data
```

Out[13]:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc
...	...	...	...	...	...	...	...
1723	low	low	5more	more	med	med	good
1724	low	low	5more	more	med	high	vgood
1725	low	low	5more	more	big	low	unacc
1726	low	low	5more	more	big	med	good
1727	low	low	5more	more	big	high	vgood

1728 rows × 7 columns

In [18]:

```
import sys

from itertools import chain, combinations
from collections import defaultdict
from optparse import OptionParser


def subsets(arr):
    """ Returns non empty subsets of arr"""
    return chain(*[combinations(arr, i + 1) for i, a in enumerate(arr)])


def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):
        """calculates the support for items in the itemSet and returns a subset
       of the itemSet each of whose elements satisfies the minimum support"""
        _itemSet = set()
        localSet = defaultdict(int)

        for item in itemSet:
                for transaction in transactionList:
                        if item.issubset(transaction):
                                freqSet[item] += 1
                                localSet[item] += 1
```

```
        for item, count in localSet.items():
                support = float(count)/len(transactionList)

                if support >= minSupport:
                        _itemSet.add(item)

        return _itemSet


def joinSet(itemSet, length):
        """Join a set with itself and returns the n-element itemsets"""
        return set([i.union(j) for i in itemSet for j in itemSet if len(i.union(j)) == l
ength])


def getItemSetTransactionList(data_iterator):
    transactionList = list()
    itemSet = set()
    for record in data_iterator:
        transaction = frozenset(record)
        transactionList.append(transaction)
        for item in transaction:
            itemSet.add(frozenset([item]))              # Generate 1-itemSets
    return itemSet, transactionList


def runApriori(data_iter, minSupport, minConfidence):
    """
    run the apriori algorithm. data_iter is a record iterator
    Return both:
     - items (tuple, support)
     - rules ((pretuple, posttuple), confidence)
    """
    itemSet, transactionList = getItemSetTransactionList(data_iter)

    freqSet = defaultdict(int)
    largeSet = dict()
    # Global dictionary which stores (key=n-itemSets,value=support)
    # which satisfy minSupport

    assocRules = dict()
    # Dictionary which stores Association Rules

    oneCSet = returnItemsWithMinSupport(itemSet,
                                        transactionList,
                                        minSupport,
                                        freqSet)

    currentLSet = oneCSet
    k = 2
    while(currentLSet != set([])):
        largeSet[k-1] = currentLSet
        currentLSet = joinSet(currentLSet, k)
        currentCSet = returnItemsWithMinSupport(currentLSet,
                                                transactionList,
                                                minSupport,
                                                freqSet)
        currentLSet = currentCSet
        k = k + 1

    def getSupport(item):
            """local function which Returns the support of an item"""
            return float(freqSet[item])/len(transactionList)

    toRetItems = []
    for key, value in largeSet.items():
        toRetItems.extend([(tuple(item), getSupport(item))
                           for item in value])

    toRetRules = []
    for key, value in list(largeSet.items())[1:]:
```

```
        for item in value:
            _subsets = map(frozenset, [x for x in subsets(item)])
            for element in _subsets:
                remain = item.difference(element)
                if len(remain) > 0:
                    confidence = getSupport(item)/getSupport(element)
                    if confidence >= minConfidence:
                        toRetRules.append(((tuple(element), tuple(remain)),
                                           confidence))
    return toRetItems, toRetRules


def printResults(items, rules):
    """prints the generated itemsets sorted by support and the confidence rules sorted by
confidence"""
    for item, support in sorted(items, key=lambda x: x[1]):
        print("item: %s , %.3f" % (str(item), support))
    print("\nRULES:")
    for rule, confidence in sorted(rules, key=lambda x: x[1]):
        pre, post = rule
        print("Rule: %s ==> %s , %.3f" % (str(pre), str(post), confidence))


def dataFromFile(fname):
        """Function which reads from the file and yields a generator"""
        with open(fname, 'rU') as file_iter:
            for line in file_iter:
                    line = line.strip().rstrip(',')                         # Remove tr
ailing comma
                    record = frozenset(line.split(','))
                    yield record

def apriori(mins=17, minc=3):
    minSupport = mins
    minConfidence = minc

    items, rules = runApriori(data, minSupport, minConfidence)

    printResults(items, rules)

apriori()
```

```
RULES:
```

In [ ]:

```

```

Question 2

In [3]:

```
df = pd.read_csv('Sales_Transactions_Dataset_Weekly.csv')
df
```

Out[3]:

	Product_Code	W0	W1	W2	W3	W4	W5
0	P1	11	12	10	8	13	12
1	P2	7	6	3	2	7	1
2	P3	7	11	8	9	10	8
3	P4	12	8	13	5	9	6
4	P5	8	5	13	11	6	7
...	...	...	...	...	...	...	...
806	P815	0	0	1	0	0	2

	Product_Code	W0	W1	W2	W3	W4	W5
807	P816	0	0	0	0	0	0
808	P817	1	0	0	0	1	1
809	P818	0	0	0	1	0	0
810	P819	0	1	0	0	0	0

811 rows × 7 columns

In [4]:

```
df.isnull().any()
```

Out[4]:

```
Product_Code    False
W0              False
W1              False
W2              False
W3              False
W4              False
W5              False
dtype: bool
```

In [5]:

```
df2 = df.copy()

zscore = StandardScaler()
df2.iloc[:,1:] = zscore.fit_transform(df.iloc[:,1:])
df2
```

Out[5]:

	Product_Code	W0	W1	W2	W3	W4	W5
0	P1	0.173919	0.228600	0.046817	-0.126810	0.261727	0.197725
1	P2	-0.157764	-0.249221	-0.490115	-0.569780	-0.196719	-0.660624
2	P3	-0.157764	0.148963	-0.106592	-0.052982	0.032504	-0.114402
3	P4	0.256839	-0.089947	0.276931	-0.348295	-0.043904	-0.270465
4	P5	-0.074843	-0.328858	0.276931	0.094675	-0.273127	-0.192434
...	...	...	...	...	...	...	...
806	P815	-0.738209	-0.727041	-0.643524	-0.717437	-0.731573	-0.582593
807	P816	-0.738209	-0.647405	-0.720229	-0.717437	-0.655165	-0.582593
808	P817	-0.655288	-0.727041	-0.720229	-0.717437	-0.655165	-0.660624
809	P818	-0.738209	-0.727041	-0.720229	-0.643608	-0.731573	-0.738656
810	P819	-0.738209	-0.647405	-0.720229	-0.717437	-0.731573	-0.738656

811 rows × 7 columns

In [6]:

```
df3 = df.copy()

minmax = MinMaxScaler(feature_range=(1,5))
df3.iloc[:,1:] = minmax.fit_transform(df.iloc[:,1:])
df3
```

Out[6]:

	Product_Code	W0	W1	W2	W3	W4	W5
0	P1	1.814815	1.905660	1.714286	1.542373	1.852459	1.923077
1	P2	1.518519	1.452830	1.214286	1.135593	1.459016	1.076923

	Product_Code	W0	W1	W2	W3	W4	W5
2	P3	1.518519	1.830189	1.571429	1.610169	1.655738	1.615385
3	P4	1.888889	1.603774	1.928571	1.338983	1.590164	1.461538
4	P5	1.592593	1.377358	1.928571	1.745763	1.393443	1.538462
...	...	...	...	...	...	...	...
806	P815	1.000000	1.000000	1.071429	1.000000	1.000000	1.153846
807	P816	1.000000	1.075472	1.000000	1.000000	1.065574	1.153846
808	P817	1.074074	1.000000	1.000000	1.000000	1.065574	1.076923
809	P818	1.000000	1.000000	1.000000	1.067797	1.000000	1.000000
810	P819	1.000000	1.075472	1.000000	1.000000	1.000000	1.000000

811 rows × 7 columns

In [ ]: