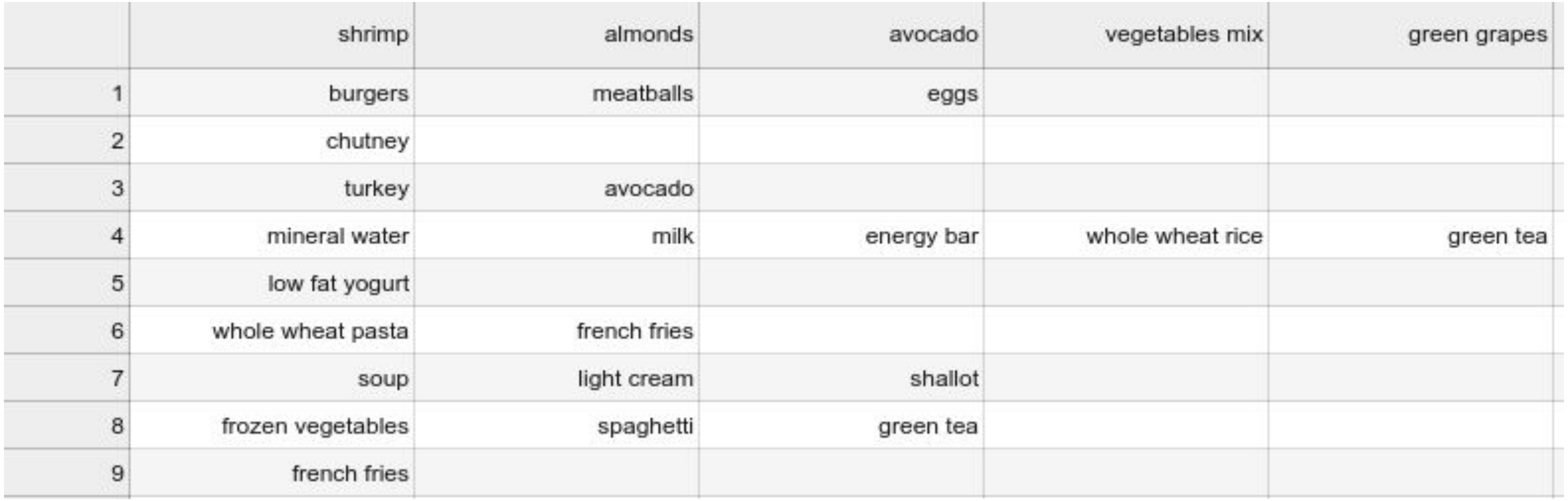
NCKU Data Mining

Project 1
學號 : F74064054
姓名 : 戴宏諺

Dataset - store_data.csv

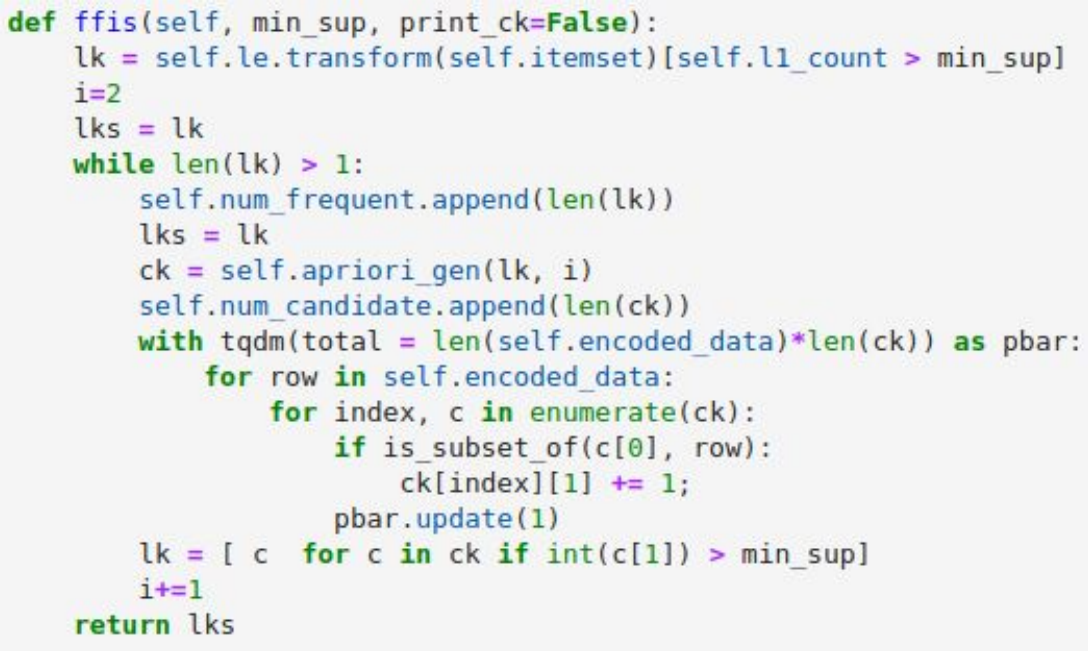
It is a dataset I found in Kaggle, which store the list in each purchase



	shrimp	almonds	avocado	vegetables mix	green grapes
1	burgers	meatballs	eggs		
2	chutney				
3	turkey	avocado			
4	mineral water	milk	energy bar	whole wheat rice	green tea
5	low fat yogurt				
6	whole wheat pasta	french fries			
7	soup	light cream	shallot		
8	frozen vegetables	spaghetti	green tea		
9	french fries				

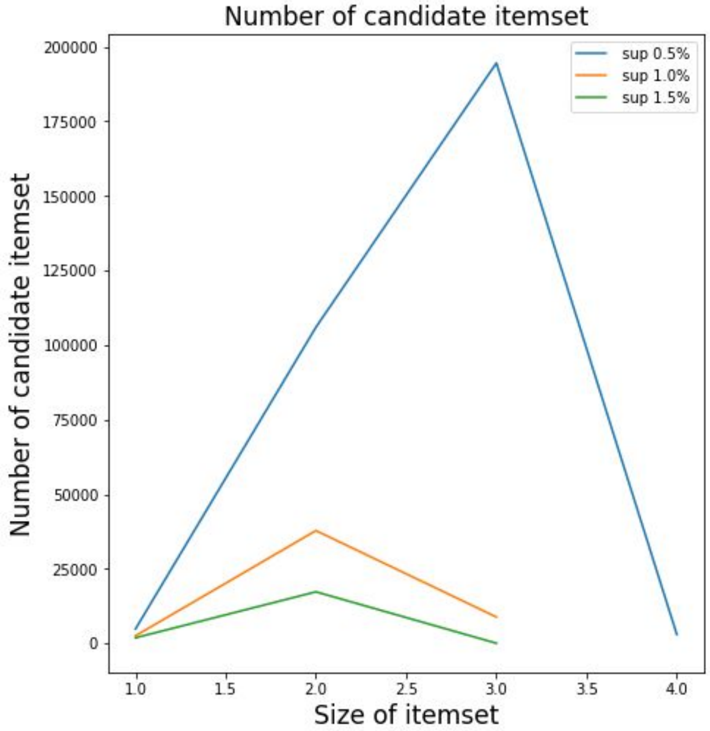
Apriori Algorithm - Implementation

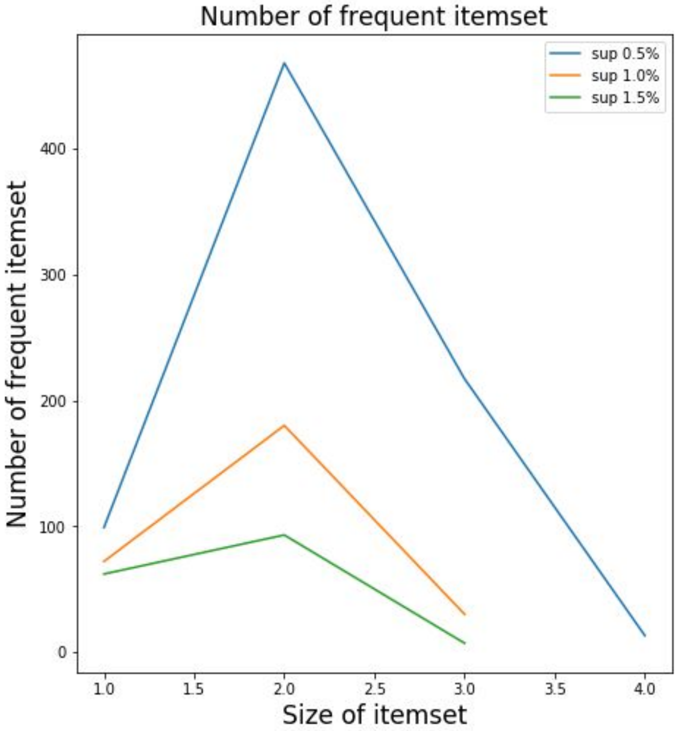
ffis (Find frequent itemset), which will implement the Apriori Algorithm.



```
def ffis(self, min_sup, print_ck=False):
    lk = self.le.transform(self.itemset)[self.l1_count > min_sup]
    i=2
    lks = lk
    while len(lk) > 1:
        self.num_frequent.append(len(lk))
        lks = lk
        ck = self.apriori_gen(lk, i)
        self.num_candidate.append(len(ck))
        with tqdm(total = len(self.encoded_data)*len(ck)) as pbar:
            for row in self.encoded_data:
                for index, c in enumerate(ck):
                    if is_subset_of(c[0], row):
                        ck[index][1] += 1;
                    pbar.update(1)
        lk = [ c  for c in ck if int(c[1]) > min_sup]
        i+=1
    return lks
```

The number of candidate and frequent





Apriori Algorithm - high conf high sup

```
['burgers' 'mineral water'] => ['eggs'],with confidence 0.541 and support 0.013

['chocolate' 'frozen vegetables'] => ['milk'],with confidence 0.565 and support 0.013

['frozen vegetables' 'milk'] => ['chocolate'],with confidence 0.59 and support 0.013

['chocolate' 'soup'] => ['mineral water'],with confidence 0.6 and support 0.012

['cooking oil' 'eggs'] => ['mineral water'],with confidence 0.733 and support 0.011
```

Apriori Algorithm - high conf low sup

```
['cooking oil' 'eggs'] => ['mineral water'],with confidence 0.7333 and support 0.011

['escalope' 'mineral water'] => ['spaghetti'],with confidence 0.5 and support 0.011

['escalope' 'spaghetti'] => ['mineral water'],with confidence 0.578 and support 0.011

['frozen vegetables' 'milk'] => ['mineral water'],with confidence 0.5 and support 0.011
```

Apriori Algorithm - low conf high sup

```
['milk'] => ['mineral water' 'chocolate'],with confidence 0.1323 and support 0.018

['chocolate' 'milk'] => ['mineral water'],with confidence 0.391 and support 0.018

['chocolate' 'mineral water'] => ['milk'],with confidence 0.305 and support 0.018

['milk' 'mineral water'] => ['chocolate'],with confidence 0.382 and support 0.018

['milk'] => ['chocolate' 'spaghetti'],with confidence 0.125 and support 0.017
```

Apriori Algorithm - low conf low sup

```
['chocolate' 'french fries'] => ['mineral water'],with confidence 0.2972 and support 0.011

['chocolate' 'mineral water'] => ['french fries'],with confidence 0.1864 and support 0.011

['frozen vegetables'] => ['chocolate' 'spaghetti'],with confidence 0.11 and support 0.011

['chocolate' 'frozen vegetables'] => ['spaghetti'],with confidence 0.478 and support 0.011
```

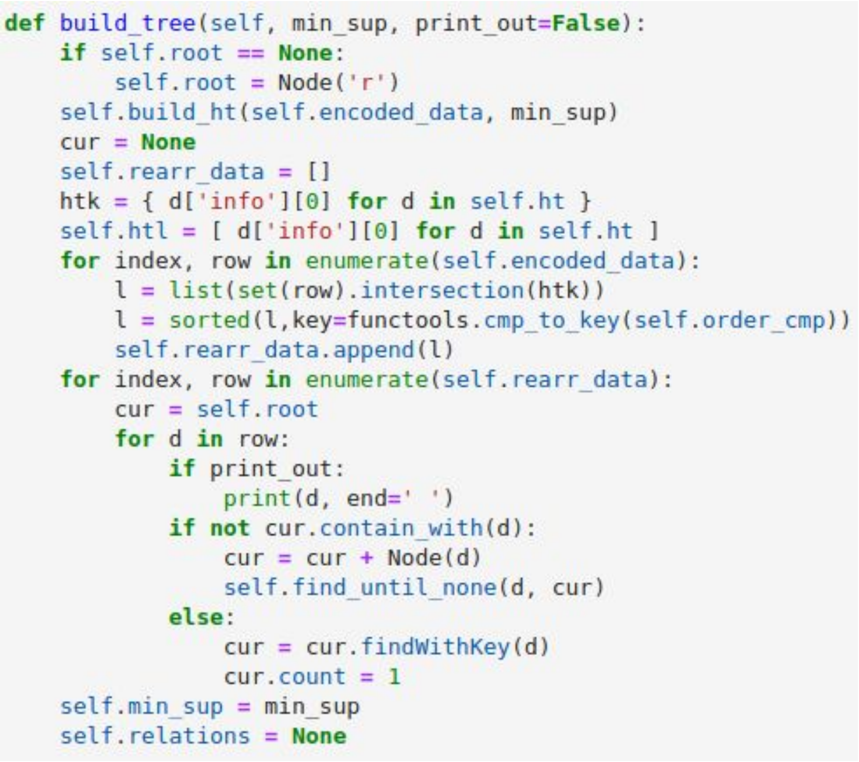
FP-growth algorithm - Implementation

# Build a frequent pattern tree

Step 1 : Build the header table and sort it with descending order.

Step 2 : According to the header table, go through the transactions and sort it which the order is identical to the header table.

Step 3: Go through the sorted transactions and build the fp-growth tree.

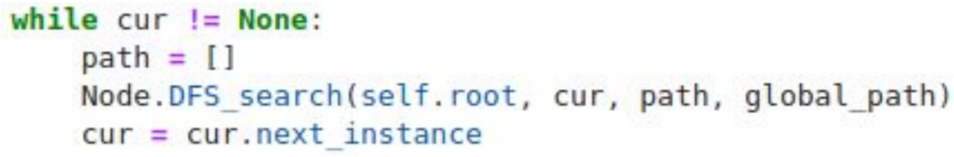


```
def build_tree(self, min_sup, print_out=False):
    if self.root == None:
        self.root = Node('r')
    self.build_ht(self.encoded_data, min_sup)
    cur = None
    self.rearr_data = []
    htk = { d['info'][0] for d in self.ht }
    self.htl = [ d['info'][0] for d in self.ht ]
    for index, row in enumerate(self.encoded_data):
        l = list(set(row).intersection(htk))
        l = sorted(l,key=functools.cmp_to_key(self.order_cmp))
        self.rearr_data.append(l)
    for index, row in enumerate(self.rearr_data):
        cur = self.root
        for d in row:
            if print_out:
                print(d, end=' ')
            if not cur.contain_with(d):
                cur = cur + Node(d)
                self.find_until_none(d, cur)
            else:
                cur = cur.findWithKey(d)
                cur.count = 1
    self.min_sup = min_sup
    self.relations = None
```

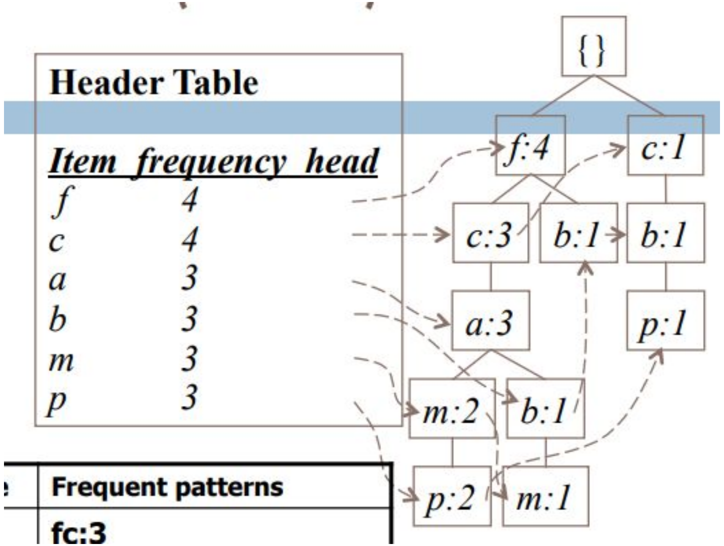
Implementation - generate cond’s pattern base

# Build a cond’s pattern base

Step 1 : According to the address of head, find the path to the Item then change to next item with same key



```
while cur != None:
    path = []
    Node.DFS_search(self.root, cur, path, global_path)
    cur = cur.next_instance
```



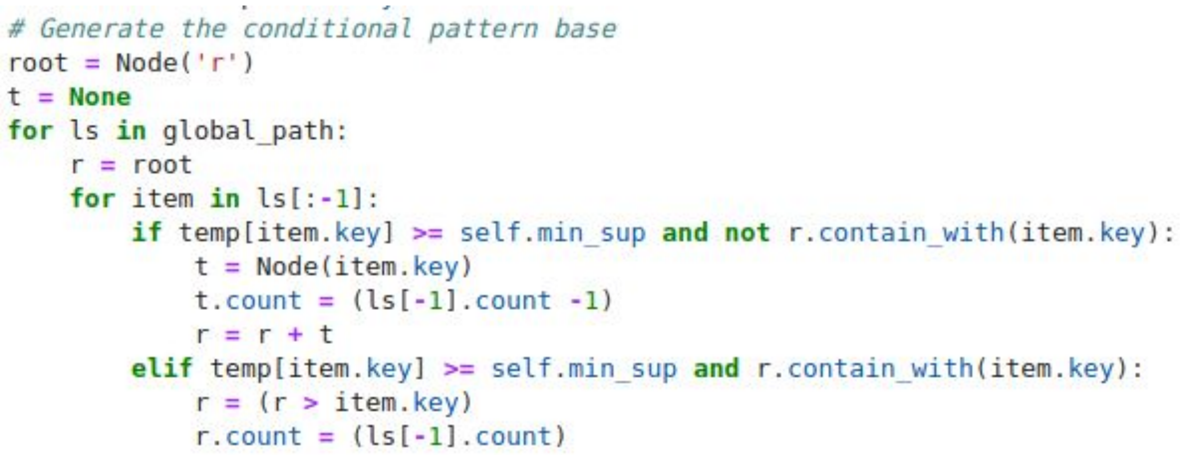
Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	

Frequent patterns
fc:3

Implementation - generate cond’s pattern base

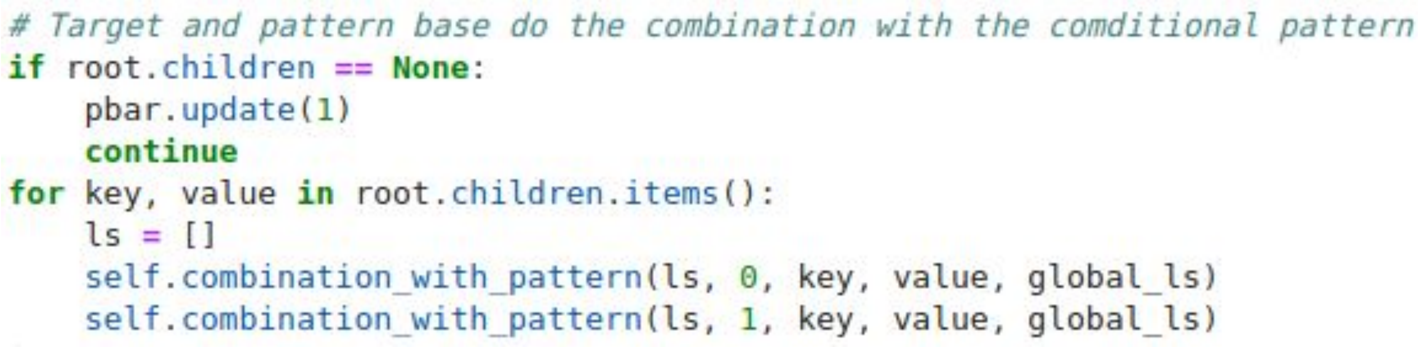
Step 2 : Construct a pattern base tree with the given path in step 1



```
# Generate the conditional pattern base
root = Node('r')
t = None
for ls in global_path:
    r = root
    for item in ls[:-1]:
        if temp[item.key] >= self.min_sup and not r.contain_with(item.key):
            t = Node(item.key)
            t.count = (ls[-1].count -1)
            r = r + t
        elif temp[item.key] >= self.min_sup and r.contain_with(item.key):
            r = (r > item.key)
            r.count = (ls[-1].count)
```

Implementation - generate cond’s pattern base

Step 3: Find all combination of pattern base tree and target item.



```
# Target and pattern base do the combination with the comditional pattern
if root.children == None:
    pbar.update(1)
    continue
for key, value in root.children.items():
    ls = []
    self.combination_with_pattern(ls, 0, key, value, global_ls)
    self.combination_with_pattern(ls, 1, key, value, global_ls)
```

FP-growth algorithm

I found some associations like this with min_support = 10

1. { mineral water , eggs , french fries }
2. { mineral water, chocolate, green tea }
3. { egg, french fries, milk }
4. { mineral water, eggs, ground beef }
5. { mineral water chocolate, french fries }
6. ...

Rules from fp-growth - high sup high conf

```
['eggs' 'cooking oil'] => ['mineral water'], with confidence 0.733 and support 0.011

['olive oil' 'soup'] => ['mineral water'], with confidence 0.733 and support 0.011

['chocolate' 'soup'] => ['mineral water'], with confidence 0.6 and support 0.012

['shrimp' 'cake'] => ['mineral water'], with confidence 0.899 and support 0.009
```

Rules from fp-growth - high sup low conf

['olive oil'] => ['mineral water' 'spaghetti'], with confidence 0.144 and support 0.011

['mineral water' 'spaghetti'] => ['olive oil'], with confidence 0.189 and support 0.011

['mineral water' 'olive oil'] => ['spaghetti'], with confidence 0.314 and support 0.011

['olive oil'] => ['mineral water' 'milk'], with confidence 0.144 and support 0.011

['mineral water' 'milk'] => ['olive oil'], with confidence 0.234 and support 0.011

['mineral water' 'olive oil'] => ['milk'], with confidence 0.314 and support 0.011

Rules from fp-growth - low sup high conf

[green tea , tomatoes] => [frozen vegetables], with confidence 0.5 and support 0.005

['spaghetti' 'whole wheat rice'] => ['chocolate'], with confidence 0.5 and support 0.007

['eggs' 'frozen smoothie'] => ['mineral water'], with confidence 0.5 and support 0.006

['chocolate' 'chicken'] => ['mineral water'], with confidence 0.5 and support 0.007

['chocolate' 'burgers'] => ['eggs'], with confidence 0.5 and support 0.008

['spaghetti' 'low fat yogurt'] => ['mineral water'], with confidence 0.7 and support 0.007

Rules from fp-growth - low sup low conf

['mineral water' 'chocolate'] => ['whole wheat rice'], with confidence 0.1186 and support 0.007

['eggs' 'chocolate'] => ['cooking oil'], with confidence 0.119 and support 0.005

['cooking oil'] => ['mineral water' 'chocolate'], with confidence 0.109 and support 0.006

['mineral water' 'chocolate'] => ['cooking oil'], with confidence 0.101 and support 0.006

['mineral water' 'eggs'] => ['frozen smoothie'], with confidence 0.1 and support 0.006