Criar a *main*

1 - Criar uma função para:

Ler o arquivo bd.csv e imprimir cada uma de suas linhas na tela

Tipo de abertura : ‘r’ por que?

Converter cada linha em uma lista de itens:

Lembrar que existe cabeçalho, como pular ele?

A primeira coluna do arquivo é o ID do usuário, como pular ela tb?

Dica: rstrip() remove o ‘\n’ do final de cada linha

**def** carregarBaseDeDados():  
 **with** open(**'bd.csv'**,**'r'**) **as** arquivo:  
 dados = []  
 cabecalho = next(arquivo)

**for** linha **in** arquivo:

itens = linha.split(**','**)  
 itens = itens[1:]  
 linhatemporaria = []

**for** item **in** itens:  
 linhatemporaria.append(item.rstrip())  
 dados.append(linhatemporaria)  
 arquivo.close()  
 **return** dados

2 - Observar que as linhas da base de dados não estão ordenadas quanto aos itens. Como ordenar? Criar uma função:

**def** ordenarBaseDeDados(basededados):  
 novaBase = []  
 **for** linha **in** basededados:  
 novalinha = sorted(linha)  
 novaBase.append(novalinha)  
 **return** novaBase

3 - identificar todos os itens da base de dados (

Para cada linha da base de dados na memória existe uma quantidade de itens que podem ser padrão:

Portanto criar uma lista e incluir TODOS os itens da base de dados SEM REPETIÇÃO

**def** geraCandidatosTamanho1(BD,suporte):  
 listaDecandidatos = []  
  
 **for** registro **in** BD: *#analisar cada linha da base de dados* **for** candidato **in** registro:  
 **if** [candidato] **not in** listaDecandidatos:  
 listaDecandidatos.append([candidato])  
  
 **return** listaDecandidatos

VALIDAÇÃO DOS CANDIDATOS:

Para cada candidato na lista de candidatos:

criar um Contador de ocorrências

verificar se o candidato está presente em cada uma das linhas da BD

DICA: usar o set para o candidato e para a linha da BD e a função issubset()

Calcular o suporte: formula e se a frequência for maior que min\_sup então é padrão. Armazena numa lista e retorna ela

**def** validaCandidatos(listaCandidatos,basededados,suporte):  
 padroes = []  
  
 **for** candidato **in** listaCandidatos:  
 contagemOcorrenciasCandidato =0  
  
 **for** registro **in** basededados:  
 s1 = set(candidato) *#set é um conjunto: como na teoria dos conjuntos da pra verificar se s1 C s2* s2 = set(registro)  
 **if** s1.issubset(s2):  
 contagemOcorrenciasCandidato += 1  
  
 frequenciaCandidato = contagemOcorrenciasCandidato/len(basededados)  
  
 **if** frequenciaCandidato >= suporte:  
 padroes.append(candidato)  
 str1 = **','**.join(candidato)  
 gravaNoArquivo(str1,frequenciaCandidato)  
 **return** padroes

PODEMOS GRAVAR OS PADRÕES QUE VAMOS ENCONTRANDO EM UM ARQUIVO>

Converter lista para uma string com separador virgula

str1 = **','**.join(candidato)

formato de saída

Lv07,Lv08:0.08163265306122448  
Lv08,Lv18:0.08163265306122448  
Lv08,Lv26:0.061224489795918366  
Lv08,Lv21:0.061224489795918366  
Lv08,Lv12:0.14285714285714285  
Lv10,Lv29:0.061224489795918366

GRAVAR TAMBÉM O SUPORTE DO PADRÃO

**def** gravaNoArquivo(padrao,suporte):  
 **with** open(**'padroesEncontrados.csv'**,**'a+'**) **as** resultado:  
 resultado.writelines(str(padrao)+**':'**+str(suporte)+**'\n'**)  
 **return**

IMPRIMIR OS PADRÕES ENCONTRADOS:

print(padroesEncontrados)

AGORA DEVE-SE COMBINAR OS PADRÕES UNITÁRIOS A FIM DE CONSTRUIR PADRÕES DE TAMANHO =2

COMO?

Uma função que receba os padrões gerados e o valor do padrão que será gerado k = 2

ORDENAR a lista dos PADRÕES que foi passada como parâmetro

Para cada um dos padrões de tamanho k-1 que já temos:

Combinar ele com os padrões a sua a frente

DICA: list(padroesPassadosParametro[i])[:k-2] # pega os k-2 elementos da lista

**def** aprioriGen(PadroesTamanhoK, k):  
 listacandidatos = []  
 PadroesTamanhoK = len(PadroesTamanhoK)  
  
 **for** i **in** range(qtdPadroesTamanhoMenor):  
 **for** j **in** range(i+1, PadroesTamanhoK):  
 L1 = list(PadroesTamanhoK[i])[:k-2]  
 print(L1)  
 L2 = list(PadroesTamanhoK[j])[:k-2]  
 print(L2)  
 L1.sort(); L2.sort()  
 **if** L1 == L2:  
 candidato = sorted(list(set(PadroesTamanhoK[i]) | set(PadroesTamanhoK[j])))  
 listacandidatos.append(candidato)  
 **return** listacandidatos

Validar os padrões de tamanho = 2

Enquanto não gerar um conjunto vazio de candidatos :

Gerar candidatos

Validar padrões

**import** csv  
**import** codigosLivros  
  
  
**def** carregarBaseDeDados():  
 **with** open(**'bd.csv'**,**'r'**) **as** arquivo:  
 dados = []  
 cabecalho = next(arquivo) *#o comando next pula o primeiro registro (O cabeçalho)* **for** linha **in** arquivo: *#para cada linha da base de dados lida temos que salvar no banco de dados da memória* itens = linha.split(**','**)  
 linhatemporaria = [i.rstrip() **for** i **in** itens[1:]]  
 dados.append(linhatemporaria)  
 arquivo.close()  
 **return** dados  
  
  
  
**def** gravaNoArquivo(padrao,suporte):  
 **with** open(**'padroesEncontrados.csv'**,**'a+'**) **as** resultado:  
 resultado.writelines(str(padrao)+**':'**+str(suporte)+**'\n'**)  
 **return  
  
  
  
def** imprimeBasedeDados(basededados):  
 **for** linha **in** basededados:  
 **for** livro **in** linha:  
 print(codigosLivros.getNomeLivro()[livro])  
  
  
**def** ordenarBaseDeDados(basededados):  
 novaBase = []  
 **for** linha **in** basededados:  
 novalinha = sorted(linha)  
 novaBase.append(novalinha)  
 **return** novaBase  
  
  
  
**def** geraCandidatosTamanho1(BD,suporte):  
 listaDecandidatos = []  
  
 **for** registro **in** BD: *#analisar cada linha da base de dados* **for** candidato **in** registro:  
 **if** [candidato] **not in** listaDecandidatos:  
 listaDecandidatos.append([candidato])  
  
 **return** listaDecandidatos  
  
  
**def** aprioriGen(PadroesTamanhoK, k):  
 listacandidatos = []  
 qtdPadroesTamanhoMenor = len(PadroesTamanhoK)  
  
 **for** i **in** range(qtdPadroesTamanhoMenor):  
 **for** j **in** range(i+1, qtdPadroesTamanhoMenor):  
 L1 = list(PadroesTamanhoK[i])[:k-2]  
 L2 = list(PadroesTamanhoK[j])[:k-2]  
 L1.sort(); L2.sort()  
 **if** L1 == L2:  
 candidato = sorted(list(set(PadroesTamanhoK[i]) | set(PadroesTamanhoK[j])))  
 listacandidatos.append(candidato)  
 **return** listacandidatos  
  
  
  
**def** validaCandidatos(listaCandidatos,basededados,suporte):  
 padroes = []  
  
 **for** candidato **in** listaCandidatos:  
 contagemOcorrenciasCandidato =0  
  
 **for** registro **in** basededados:  
 s1 = set(candidato) *#set é um conjunto: como na teoria dos conjuntos da pra verificar se s1 C s2* s2 = set(registro)  
 **if** s1.issubset(s2):  
 contagemOcorrenciasCandidato += 1  
  
 frequenciaCandidato = contagemOcorrenciasCandidato/len(basededados)  
  
 **if** frequenciaCandidato >= suporte:  
 padroes.append(candidato)  
 str1 = **','**.join(candidato)  
 gravaNoArquivo(str1,frequenciaCandidato)  
 **return** padroes  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 padroesEncontrados = []  
 suporte = 0.05  
  
 print(**'Carregando a base de dados'**)  
 basededados = carregarBaseDeDados()  
 basededados = ordenarBaseDeDados(basededados)  
  
  
 print(**'Identificando os padrões de tamanho = 1 ...'**)  
 candidatosTamanho1 = geraCandidatosTamanho1(basededados,suporte)  
 padroesEncontrados = validaCandidatos(candidatosTamanho1,basededados,suporte)  
 print(padroesEncontrados)  
  
 print(**'gerando candidatos a padrões de tamanho = 2'**)  
 candidatosTamanho2 = aprioriGen(padroesEncontrados,2)  
 print(**'Validando os candidatos como padrões'**)  
 padroesTamanho2 = validaCandidatos(candidatosTamanho2,basededados,suporte)  
 print(padroesTamanho2)  
  
 padroesEncontrados += padroesTamanho2  
 k = 3  
 print(**'gerando candidatos a padrões de tamanho = %s'** %k)  
 listaDeCandidatos = aprioriGen(padroesTamanho2,k)  
  
 **while** listaDeCandidatos != []:  
 print(**'Validando os candidatos como padrões de tamanho = %s'** %k)  
 padroes = validaCandidatos(listaDeCandidatos,basededados,suporte)  
 print(padroes)  
 k += 1  
 print(**'gerando candidatos a padrões de tamanho = %s'** % k)  
 listaDeCandidatos = aprioriGen(padroes, k)  
 **if** listaDeCandidatos == []:  
 print(**'Nenhum novo candidato pôde ser gerado, O algoritmo chegou ao final de sua execução'**)  
 **break**