Lista 7

Link para o repositório:
https://github.com/AHChaves/Aprendizado-de-Maquina

Q1- A partir dos dados informados temos: 3 itens no itemset 1; 3 itens no itemset 2; 1 itens no itemset 3 e 7 regras válidas

itemset 1:
Café 3/10
Pão 5/10
Manteiga 5/10

itemset 2:
Café e Pão 3/10
Café e Manteiga 3/10
Manteiga e Pão 4/10

itemset 3:
Café, Pão e Manteiga 3/10

Regras:
Café → Pão 3/3 = 1
Pão → Café 3/5 = 0.6 **(não válida)**

Café → Manteiga 3/3 = 1
Manteiga → Café 3/5 = 0.6 **(não válida)**

Manteiga → Pão 4/5 = 0.8
Pão → Manteiga 4/5 = 0.8

Café e Pão → Manteiga 3/3 = 1
Café e Manteiga → Pão 3/3 = 1
Manteiga e Pão → Café 3/4 = 0.75 **(não válida)**
Café → Manteiga e Pão 3/3 = 1
Pão → Manteiga e Café 3/5 = 0.6 **(não válida)**

Manteiga → Pão e Café 3/5 = 0.6 **(não válida)**

Q2- Após a execução do código dentro da base e da remoção dos dados ausentes nas regras por meio do seguinte código:

```
transacoes = []
for i in range(len(base)):
    transacoes.append([ str(base.values[i, j]) for j in range(base.shape[1]) if not pd.isna(base.values[i, j])])
```

✓ 0.0s

pudemos observar resultados idênticos a da questão acima, confirmando assim os resultados.

	Lhs	Rhs	# suporte	# confianca	# lift
0	['Cafe']	['Manteiga']	0.3	1.0	2.0
1	['Cafe']	['Pao']	0.3	1.0	2.0
2	['Manteiga']	['Pao']	0.4	0.8	1.6
3	['Pao']	['Manteiga']	0.4	0.8	1.6
4	['Cafe']	['Manteiga', 'Pao']	0.3	1.0	2.5
5	['Manteiga', 'Cafe']	['Pao']	0.3	1.0	2.0
6	['Cafe', 'Pao']	['Manteiga']	0.3	1.0	2.0

Q3- Para visualizar os itemsets foi utilizado o código abaixo, que retira da saída do algoritmo apriori e formata para um dataset

```
itemsets_com_suporte = []
for resultado in saida:
    itemset = list(resultado[0])
    suporte = resultado[1]
    itemsets_com_suporte.append((itemset, suporte))

df_itemsets = pd.DataFrame(itemsets_com_suporte, columns=['Itemset', 'Suporte'])

df_itemsets = df_itemsets.sort_values(by='Suporte', ascending=False)

print("Itemsets gerados com seus respectivos suportes:")
print(df_itemsets)
```

✓ 0.0s Open 'df_itemsets' in Data Wrangler

Foi possível visualizar os seguintes itens. A falta de itens, em comparação com a questão 1, se deve a forma como o algoritmo escolhe as saídas já filtrando pelo suporte e confiança.

	Itemset	# Suporte
2	['Manteiga', 'Pao']	0.4
0	['Manteiga', 'Cafe']	0.3
1	['Cafe', 'Pao']	0.3
3	['Manteiga', 'Cafe', 'Pao']	0.3

Caso queiramos visualizar de uma forma similar a questão 1, basta reduzir a confiança para zero e teremos o seguinte resultado.

	Itemset	# Suporte
0	['Cafe']	0.3
1	['Manteiga']	0.5
2	['Pao']	0.5
3	['Manteiga', 'Cafe']	0.3
4	['Cafe', 'Pao']	0.3
5	['Manteiga', 'Pao']	0.4
6	['Manteiga', 'Cafe', 'Pao']	0.3

Q4- Fazendo a associação pela regra, quando não há X tem Y, temos o seguinte código modificado

```
todos_itens = set()
for transacao in transacoes:
    for item in transacao:
        todos_itens.add(item)
todos_itens = list(todos_itens)
```
✓ 0.0s

nesse primeiro trecho foi criado um vetor com todos os possíveis valores do dataset. Neste próximo trecho é realizado uma modificação nos valores, a adição do NÃO_item, que indica a falta de presença daquele item na instância.

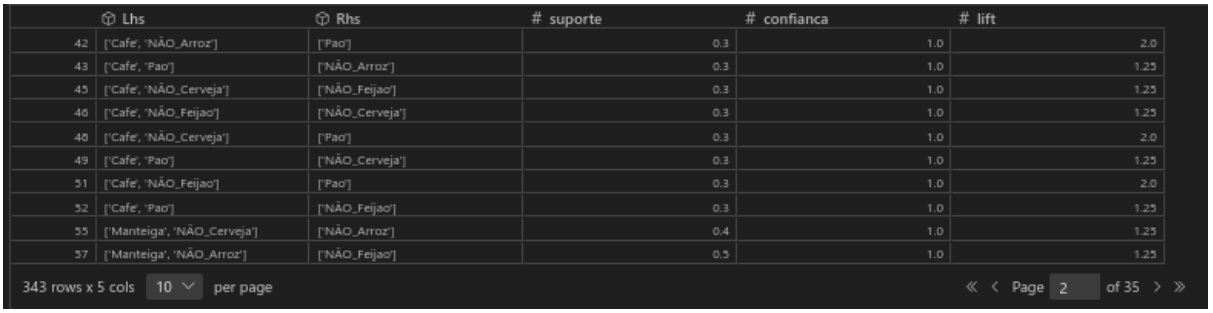
```
transacoes_expandidas = []
for transacao in transacoes:
    transacao_expandida = transacao.copy()
    for item in todos_itens:
        if item not in transacao:
            transacao_expandida.append(f"NÃO_{item}")
    transacoes_expandidas.append(transacao_expandida)
```
✓ 0.0s

Aplicando o algoritmo de associação pode-se notar um grande aumento no número de itens total, tendo 117 itens distribuídos pelos itemsets

```
regras = apriori(transacoes_expandidas, min_support=0.3, min_confidence=0.8)
saida = list(regras)
len(saida)
```
✓ 0.0s

```
117
```

Observando agora as regras, foi possível observar um aumento nelas também, de 7 regras passamos a ter 343.



	Lhs	Rhs	# suporte	# confianca	# lift
42	['Cafe', 'NÃO_Arroz']	['Pao']	0.3	1.0	2.0
43	['Cafe', 'Pao']	['NÃO_Arroz']	0.3	1.0	1.25
45	['Cafe', 'NÃO_Cerveja']	['NÃO_Feijao']	0.3	1.0	1.25
46	['Cafe', 'NÃO_Feijao']	['NÃO_Cerveja']	0.3	1.0	1.25
48	['Cafe', 'NÃO_Cerveja']	['Pao']	0.3	1.0	2.0
49	['Cafe', 'Pao']	['NÃO_Cerveja']	0.3	1.0	1.25
51	['Cafe', 'NÃO_Feijao']	['Pao']	0.3	1.0	2.0
52	['Cafe', 'Pao']	['NÃO_Feijao']	0.3	1.0	1.25
55	['Manteiga', 'NÃO_Cerveja']	['NÃO_Arroz']	0.4	1.0	1.25
57	['Manteiga', 'NÃO_Arroz']	['NÃO_Feijao']	0.5	1.0	1.25

343 rows x 5 cols 10 per page « ‹ Page 2 of 35 › »

Por meio da imagem acima podemos notar regras como: “quem leva Café e não leva arroz leva pão”.

Q5- A biblioteca MLxtend realiza as mesmas funções que a biblioteca utilizada anteriormente, no entanto a aplicação difere. Na minha concepção ela se torna mais fácil, deixando o código mais limpo, simples e compreensível. Devido a isso considerei ela melhor que a anterior.

```
te = TransactionEncoder()
te_ary = te.fit(transacoes).transform(transacoes)
df = pd.DataFrame(te_ary, columns=te.columns_)
```
✓ 0.0s

Uma adição feita por essa biblioteca são as funções de codificação, que automaticamente transforma as instâncias em verdadeiro ou falso, para o caso de presença daquele atributo.

```
frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)
```
✓ 0.0s

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.8)
```
✓ 0.0s

Nesse algoritmo nós separamos os itemsets por meio da função apriori e, após isso, utilizamos esses itemsets para gerar regras por meio da função ‘association_rules’.

Q6- A Leverage analisa quanto dois itens aparecem em relação ao que realmente aconteceu, analisando também a independência dos dois itens. Ele busca identificar relações fortes em dados esparsos e evitar regras óbvias. Ele se baseia na regra da adição da probabilidade.

Leverage = P(dos dois itens) - [P(item1) x P(item2)]

ex: Leverage(Café e Pão) = 0.3 - (0.3 * 0.5) = 0.3 - 0.15 = 0.15

Outra métrica é a Certainty que avalia o quão certa esta uma regra. Os valores variam de -1 a 1, sendo 1 a certeza máxima e -1 a certeza mínima.

Certainty(regra) =(Confiança - P(item2))/(1 - P(item2)) se Confiança > P(Y)
Certainty(regra) =(Confiança - P(item2))/(P(item2)) se Confiança < P(Y)

item2 sendo aquele após a seta(→).

ex:

Certainty(Café → Pão) = (1 - 0.5)/(1 - 0.5) = 0.5/0.5 = 1

A biblioteca MLxtend já realiza o cálculo dessas métricas automaticamente, então basta imprimir elas para ver os resultados.

```
print("\nRegras de associação:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'leverage', 'certainty']])
```

✓ 0.0s

Regras de associação:

	antecedents	consequents	support	confidence	leverage	certainty
0	(Cafe)	(Manteiga)	0.3	1.0	0.15	1.0
1	(Cafe)	(Pao)	0.3	1.0	0.15	1.0
2	(Pao)	(Manteiga)	0.4	0.8	0.15	0.6
3	(Manteiga)	(Pao)	0.4	0.8	0.15	0.6
4	(Pao, Cafe)	(Manteiga)	0.3	1.0	0.15	1.0
5	(Cafe, Manteiga)	(Pao)	0.3	1.0	0.15	1.0
6	(Cafe)	(Pao, Manteiga)	0.3	1.0	0.18	1.0

Q7- O artigo detalha os métodos de visualização para mineração de regras de associação (ARM), explorando sua evolução, características e aplicações. Nele é destacado a importância da visualização no processo de ARM, que envolve pré-processamento, mineração de regras e pós-processamento. A visualização é extremamente importante para tornar os resultados compreensíveis, principalmente diante de um grande volume de regras geradas, muitas vezes não interpretáveis para as pessoas leigas no assunto.

O estudo traça a história da ARM, desde o algoritmo Apriori até métodos atuais baseados em abordagens estocásticas. Os autores enfatizam a relevância da ARM em domínios como análise de mercado, diagnóstico médico e sequências de proteínas. É abordado também a complexidade da pipeline de ARM, que envolve limpeza de dados, mineração e pós-processamento, onde a visualização desempenha um papel crucial.