Questão 2:

```
# Instalação da biblioteca necessária

# $ pip install apyori

# Importações

import pandas as pd

from apyori import apriori

# Leitura do arquivo .csv

dados = pd.read_csv('supermercado.csv', sep=';', encoding='utf-8', header=None)

print("Dados carregados:")

print(dados)

print("Dimensões (linhas x colunas):", dados.shape)

print("// ========================== //")



# Extraindo os nomes dos produtos e transações

produtos = dados.iloc[0, 1:].values.tolist()

registros = dados.iloc[1:].reset_index(drop=True)

print("Produtos disponíveis:")

print(produtos)

print("Transações registradas:")

print(registros)

print("// ========================== //")

# Criando lista de listas para as transações

lista_transacoes = []

for _, linha in registros.iterrows():

    itens_presente = [produtos[i] for i in range(len(produtos)) if linha[i + 1] == 'Sim']
```

```
    lista_transacoes.append(itens_presente)

lista_transacoes.sort(key=lambda x: len(x))

print("Transações processadas:")

for trans in lista_transacoes:

    print(trans)

print("// ========================== //")

# Aplicação do algoritmo Apriori

regras_encontradas = apriori(lista_transacoes, min_support=0.3, min_confidence=0.8)

resultado = list(regras_encontradas)



print("Total de regras encontradas:", len(resultado))

print("Regras geradas:")

for r in resultado:

    print(r)

print("// ========================== //")

# Organização das regras em DataFrame

col_antecedente = []

col_consequente = []

col_suporte = []

col_confianca = []

col_lift = []



for regra in resultado:

    suporte_regra = regra.support

    estatisticas = regra.ordered_statistics
```

```
    for estatistica in estatisticas:

        lhs = list(estatistica.items_base)

        rhs = list(estatistica.items_add)

        conf = estatistica.confidence

        elev = estatistica.lift



        if not lhs or not rhs or 'nan' in lhs or 'nan' in rhs:

            continue



        col_antecedente.append(lhs)

        col_consequente.append(rhs)

        col_suporte.append(suporte_regra)

        col_confianca.append(conf)

        col_lift.append(elev)



tabela_regras = pd.DataFrame({

    'Antecedente': col_antecedente,

    'Consequente': col_consequente,

    'Suporte': col_suporte,

    'Confiança': col_confianca,

    'Lift': col_lift

})

print("DataFrame com regras formatadas:")

print(tabela_regras)

print("// ========================== //")
```

```
# Exibindo regras ordenadas por Lift

print("Regras ordenadas por 'Lift':")

print(tabela_regras.sort_values(by='Lift', ascending=False))
```

==========

Questão 03:

```
# Instalação da biblioteca necessária

# $ pip install apyori

# Bibliotecas utilizadas

import pandas as pd

from itertools import combinations

from collections import Counter

# Carregamento do arquivo CSV

tabela = pd.read_csv('supermercado.csv', sep=';', encoding='utf-8', header=None)

print("Conteúdo da Tabela:")

print(tabela)

print("Tamanho da Tabela (linhas x colunas):", tabela.shape)

print("// ============================== //")




# Extraindo os itens e transações

nomes_itens = tabela.iloc[0, 1:].tolist()

linhas_transacoes = tabela.iloc[1:].reset_index(drop=True)

print("Lista de Itens:")

print(nomes_itens)
```

```
print("Transações Registradas:")

print(linhas_transacoes)

print("// ============================== //")

quantidade_transacoes = len(linhas_transacoes)



# Geração de todas as combinações possíveis

conjuntos_gerados = []

for _, linha in linhas_transacoes.iterrows():

    presentes = [nomes_itens[i] for i in range(len(nomes_itens)) if linha[i + 1] == 'Sim']

    for tam in range(1, len(presentes) + 1):

        combinacoes = combinations(sorted(presentes), tam)

        conjuntos_gerados.extend(combinacoes)

# Contabilização dos itemsets

frequencia_itemsets = Counter(conjuntos_gerados)



# Exibição do suporte dos itemsets

print("Suporte por Conjunto de Itens:")

nivel_atual = 0

for conjunto, ocorrencias in sorted(frequencia_itemsets.items(), key=lambda x: (len(x[0]),
x[0])):

    tamanho_atual = len(conjunto)

    if tamanho_atual != nivel_atual:

        print(f"\nConjuntos com {tamanho_atual} item(ns):")

        nivel_atual = tamanho_atual

    suporte = ocorrencias / quantidade_transacoes

    print(f"  {list(conjunto)} -> suporte: {suporte:.3f}
```

```
({ocorrencias}/{quantidade_transacoes})")
```

==========

Questão 04:

```
# Instalação do pacote necessário:

# $ pip install apyori

# Bibliotecas

import pandas as pd

from apyori import apriori

# Leitura do arquivo CSV

dados = pd.read_csv('supermercado.csv', sep=';', encoding='utf-8', header=None)

print("Conteúdo original:")

print(dados)

print("Dimensões (linhas x colunas):", dados.shape)

print("// ============================ //")



# Extração do cabeçalho e das transações

colunas = dados.iloc[0, 1:].tolist()

registros = dados.iloc[1:].reset_index(drop=True)

print("Itens identificados:")

print(colunas)

print("Registros de compra:")

print(registros)

print("// ============================ //")

# Preparando os dados para o algoritmo Apriori

lista_transacoes = []
```

```
for _, linha in registros.iterrows():
    itens_negados = [colunas[i] for i in range(len(colunas)) if linha[i + 1] == 'Não']
    lista_transacoes.append(itens_negados)
# Ordenar por tamanho das transações (opcional)
lista_transacoes.sort(key=len)
print("Transações formatadas:")
for t in lista_transacoes:
    print(t)
print("// ============================ //")
# Execução do algoritmo Apriori
resultados = apriori(lista_transacoes, min_support=0.3, min_confidence=0.8)
regras_brutas = list(resultados)
print("Total de regras encontradas:", len(regras_brutas))
print("Regras encontradas:")
for r in regras_brutas:
    print(r)
print("// ============================ //")
# Processamento das regras para exibição em DataFrame
antecedentes, consequentes, suportes, confiancas, lifts = [], [], [], [], []
for regra in regras_brutas:
    suporte_regra = regra.support
    detalhes = regra.ordered_statistics
    for d in detalhes:
        ant = list(d.items_base)
        cons = list(d.items_add)
```

```
        conf = d.confidence
        lf = d.lift


        if not ant or not cons: continue
        if 'nan' in ant or 'nan' in cons: continue


        antecedentes.append(ant)
        consequentes.append(cons)
        suportes.append(suporte_regra)
        confiancas.append(conf)
        lifts.append(lf)


# Criação do DataFrame final
tabela_regras = pd.DataFrame({
    'Antecedente': antecedentes,
    'Consequente': consequentes,
    'Suporte': suportes,
    'Confiança': confiancas,
    'Lift': lifts
})
print("Tabela de Regras Geradas:")
print(tabela_regras)
print("// ============================ //")


# Ordenação pela métrica lift
```

```
print("Tabela de Regras Ordenada por Lift:")

ordenadas = tabela_regras.sort_values(by='Lift', ascending=False)

print(ordenadas)
```

==========

Questão 05:

```
# Instalação da dependência:

# $ pip install mlxtend

# Docs:

# https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/

# https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/

# Bibliotecas

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent_patterns import apriori, association_rules

# Leitura do arquivo CSV

dados_brutos = pd.read_csv('supermercado.csv', sep=';', encoding='utf-8')

print("Conteúdo inicial do DataFrame:")

print(dados_brutos)

print("Tamanho da tabela:", dados_brutos.shape)

print("// ============================== //")
```

```
# Conversão para lista de transações

colunas = list(dados_brutos.columns)

compras_formatadas = []

for _, linha in dados_brutos.iterrows():

    itens_confirmados = [colunas[i] for i in range(len(colunas)) if linha[i] == 'Sim']

    compras_formatadas.append(itens_confirmados)

print("Transações identificadas:")

print(compras_formatadas)

print("// ============================== //")

# Transformação com TransactionEncoder

encoder = TransactionEncoder()

dados_codificados = encoder.fit_transform(compras_formatadas)

df_binario = pd.DataFrame(dados_codificados, columns=encoder.columns_)

print("DataFrame binário para Apriori:")

print(df_binario)

print("// ============================== //")

# Execução do Apriori

itemsets_freq = apriori(df_binario, min_support=0.3, use_colnames=True)

itemsets_freq['Tamanho'] = itemsets_freq['itemsets'].apply(len)

print("Itemsets frequentes encontrados:")

print(itemsets_freq)

print("// ============================== //")

# Geração de regras de associação

regras_geradas = association_rules(itemsets_freq, metric="confidence", min_threshold=0.8)

print("Regras derivadas:")
```

```
print(regras_geradas)

print("// ============================== //")



# Impressão das regras de forma legível

print("Regras formatadas (quem compra leva):")

for _, linha in regras_geradas.iterrows():

    ant = [str(x) for x in linha['antecedents']]

    cons = [str(x) for x in linha['consequents']]

    print(f"Quem compra {', '.join(ant)} também compra {', '.join(cons)}")
```

Questão 06:

O texto elaborado por Iztok Fister Jr. e sua equipe oferece uma análise abrangente e sistemática dos métodos de representação visual usados em regras de associação, uma técnica chave na área de mineração de dados. O principal objetivo da pesquisa é destacar como as diversas abordagens de visualização auxiliam na compreensão e interpretação das regras formuladas pelos algoritmos de mineração, principalmente em contextos de elevado volume de dados.A organização do texto apresenta uma linha clara e educativa, começando com uma contextualização do surgimento da mineração de regras de associação (ARM), suas aplicações mais relevantes e a evolução ao longo do tempo — desde algoritmos tradicionais como Apriori até as abordagens mais novas que utilizam métodos estocásticos e sequenciais.Um dos principais aspectos do trabalho é a taxonomia minuciosa das técnicas de visualização, que são divididas em duas categorias principais:Técnicas convencionais, como gráficos de dispersão, redes, matrizes, gráficos mosaico e gráficos de dois andares;Técnicas inovadoras, como diagramas de Ishikawa (fishbone), representações moleculares, redes conceituais, mapas de metrô, diagramas de Sankey, gráficos de fita e gráficos baseados em glyphs.Cada técnica é examinada quanto a suas características, benefícios, desvantagens e aplicações. O artigo ainda define critérios de comparação, como o tipo de atributos que cada forma de visualização pode representar (categóricos, numéricos ou binários), o número de métricas relevantes que são apresentadas (como suporte, confiança e lift) e a habilidade de lidar com grandes conjuntos de regras.Adicionalmente, os autores fazem uma ligação dessa análise com o campo emergente da Inteligência Artificial Explicável (XAI), ressaltando a relevância da visualização como um recurso para tornar os modelos de aprendizado automático mais acessíveis e compreensíveis.

O estudo também sugere direções a serem seguidas no futuro, destacando desafios que permanecem, como a escalabilidade das visualizações, a integração com métodos interativos e personalizáveis, e a necessidade de uma maior padronização e facilidade de uso dos softwares disponíveis.