

ANÁLISE DE VENDAS

Proposta de Projeto - Vaga Analista de Dados

Rafaela Pacheco de Oliveira

Proposta

Encontre itens com potencial de venda conjunta.

- Ferramentas utilizadas:
 - Fonte de dados Access;
 - Linguagem Python, pacote mlxtend;
 - Visualização software Power BI.

```
mirror_mod = modifier_ob.  
#set mirror object to mirror  
mirror_mod.mirror_object =  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
  
#selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
  
print("please select exactly  
  
-- OPERATOR CLASSES --  
  
types.Operator):  
X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
  
context):  
context.active_object is not
```

Estruturação do script

- ▶ O código foi escrito orientado a objetos, cada uma das classes possui uma função específica, tornando o código mais modular e facilitando a compreensão e manutenção.



- Importação de bibliotecas:
As bibliotecas necessárias são importadas, incluindo pyodbc, os, pandas e algumas funções específicas do mlxtend para mineração de regras de associação.

```
1 import pyodbc
2 import os
3 import pandas as pd
4 from mlxtend.frequent_patterns import apriori
5 from mlxtend.frequent_patterns import association_rules
6
7 import warnings
8 warnings.filterwarnings('ignore')
9
```

- Definição da classe DatabaseConnection: Essa classe estabelece a conexão com o banco de dados Access. O construtor cria a conexão usando o driver ODBC do Access e fornece métodos para executar consultas SQL e obter Dataframes a partir dos resultados.

```
10
11 class DatabaseConnection:
12     def __init__(self):
13         print("Criando a conexão...")
14         current_dir = os.path.dirname(os.path.realpath("foo"))
15         db_file = "compras2014.mdb"
16         db_path = os.path.join(current_dir, db_file)
17         self.conn = pyodbc.connect(r'Driver={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=' + db_path)
18         self.cursor = self.conn.cursor()
19
20     def close_connection(self):
21         print("Fechando a conexão ao banco Access...")
22         self.conn.close()
23
24     def get_dataframe_from_sql(self, query):
25         return pd.read_sql(query, self.conn)
26
```

- Definição da classe DataPreprocessing: Essa classe é responsável pelo pré-processamento dos dados. O método "preprocess_data" executa consultas SQL para obter os dados brutos do banco de dados Access. Em seguida, ele realiza o tratamento dos dados, como substituições de valores e mesclagem de Dataframes;
- Foram criados dois Dataframes sendo um (df_marca) para composição do Diagrama de Venn no visual do Power BI, e outro (df) para utilização no algoritmo Apriori.

```
def preprocess_data(self):
    print("Variáveis com as queries...")
    consulta_transacoes = "SELECT * FROM transacoes"
    consulta_itens = "SELECT * FROM itens"
    consulta_itemtransacao = "SELECT * FROM itemtransacao"

    print("Gerando os Dataframes...")
    df_transacoes = self.conn.get_dataframe_from_sql(consulta_transacoes)
    df_itens = self.conn.get_dataframe_from_sql(consulta_itens)
    df_itemtransacao = self.conn.get_dataframe_from_sql(consulta_itemtransacao)

    print("Tratamento das bases...")
    df_itens = df_itens.replace({'limao': 'limão', 'refrigerante': 'refrigerante', 'limao': 'limão', 'sabao em po': 'sabão em pó'})

    for y in ['descrição', 'marca', 'tipo']:
        df_itens[y] = df_itens[y].apply(lambda x: x.title())

    df_itemtransacao = pd.merge(df_itemtransacao, df_itens, how='left', left_on=['item'], right_on=['codItem'])

    df_marca = df_itemtransacao.copy()[['marca']].drop_duplicates()
    df_marca['codMarca'] = range(len(df_marca))

    df_itemtransacao = pd.merge(df_itemtransacao, df_marca, how='left', on=['marca'])
```

```
df = pd.pivot_table(df_itemtransacao,
                    index=['IDTransação'],
                    columns=['descrição'],
                    values=['item'],
                    aggfunc={'item': lambda x: len(x.unique())},
                    fill_value=0)

df = df.droplevel(0, axis=1)

df_marca = pd.pivot_table(df_itemtransacao,
                        index=['IDTransação'],
                        columns=['marca'],
                        values=['codMarca'],
                        aggfunc={'codMarca': lambda x: len(x.unique())},
                        fill_value=0)

df_marca = df_marca.droplevel(0, axis=1)

return df_transacoes, df_itens, df_itemtransacao, df, df_marca
```

- Definição da classe AssociationAnalysis: Essa classe realiza a análise de regras de associação. O método "generate_association_rules" aplica o algoritmo Apriori aos dados pré-processados para extrair as regras de associação.
- O suporte mínimo foi definido em 10% devido ao tamanho da base de dados, buscando os itens com maior repetibilidade.
- Para gerar as regras de associação foi utilizado o association_rules com uma confiança mínima escolhida de 50%.
- Por fim foram filtrados os resultados com lift maior que 1, para removermos produtos que pudessem ser muito vendidos indiferente de estarem associado aos demais.

```
77 class AssociationAnalysis:
78     def generate_association_rules(self, df):
79         print("Regras de Associação..")
80         frequent_items = apriori(df, min_support=0.1, use_colnames=True)
81
82         rules = association_rules(frequent_items, metric="confidence", min_threshold=0.5)
83         rules = rules[(rules['lift'] > 1) & (rules['zhangs_metric'] > 0.5)]
84
85         apriori_rules = rules
86         apriori_rules['lhs_items'] = apriori_rules['antecedents'].apply(lambda x: len(x))
87         apriori_rules[apriori_rules['lhs_items'] > 1].sort_values('lift', ascending=False).head()
88         apriori_rules['antecedents_'] = apriori_rules['antecedents'].apply(lambda a: ', '.join(list(a)))
89         apriori_rules['consequents_'] = apriori_rules['consequents'].apply(lambda a: ', '.join(list(a)))
90
91         apriori_rules = apriori_rules[['antecedents_', 'consequents_', 'support', 'confidence', 'lift',
92                                         'zhangs_metric', 'lhs_items']]
93
94         support_table = pd.pivot_table(apriori_rules,
95                                         index='consequents_',
96                                         columns='antecedents_',
97                                         values='support')
98
99         return apriori_rules, support_table, frequent_items, rules
```

- Definição da classe `DataAnalysis`: Essa classe gerencia as etapas da análise de dados. Ela cria uma instância da classe `DatabaseConnection`, `DataPreprocessing` e `AssociationAnalysis` para executar as respectivas etapas. O método `run_analysis` executa todo o fluxo de análise e retorna os resultados.

```
101
102 class DataAnalysis:
103     def __init__(self):
104         self.db_conn = DatabaseConnection()
105         self.data_preprocessing = DataPreprocessing(self.db_conn)
106         self.association_analysis = AssociationAnalysis()
107
108     def run_analysis(self):
109         df_transacoes, df_itens, df_itemtransacao, df, df_marca = self.data_preprocessing.preprocess_data()
110         apriori_rules, support_table, frequent_items, rules = self.association_analysis.generate_association_rules(df)
111         self.db_conn.close_connection()
112         return df_transacoes, df_itens, df_itemtransacao, df, df_marca, apriori_rules, support_table, frequent_items, rules
113
```



- Criação de uma instância da classe `DataAnalysis`: Uma instância da classe `DataAnalysis` é criada.
- Execução da análise de dados: A análise de dados é executada chamando o método `run_analysis` na instância de `DataAnalysis`. Os resultados são atribuídos a várias variáveis para posterior uso.


```
114 |  
115 data_analysis = DataAnalysis()  
116 df_transacoes, df_itens, df_itemtransacao, df, df_marca, apriori_rules, support_table, frequent_items, rules = data_analysis.run_analysis()
```


Modelagem Power BI


- A finalização do código gera as bases necessárias para o projeto, que foram importadas no Access.


Nome


 apriori_rules

 compras2014

 df_itemtransacao

 df_itens

 df_marca

 df_transacoes

Banco de dados do Microsoft Access

☒ Básico ☐ Avançadas

Caminho do arquivo

Abrir arquivo como

Banco de dados do Access

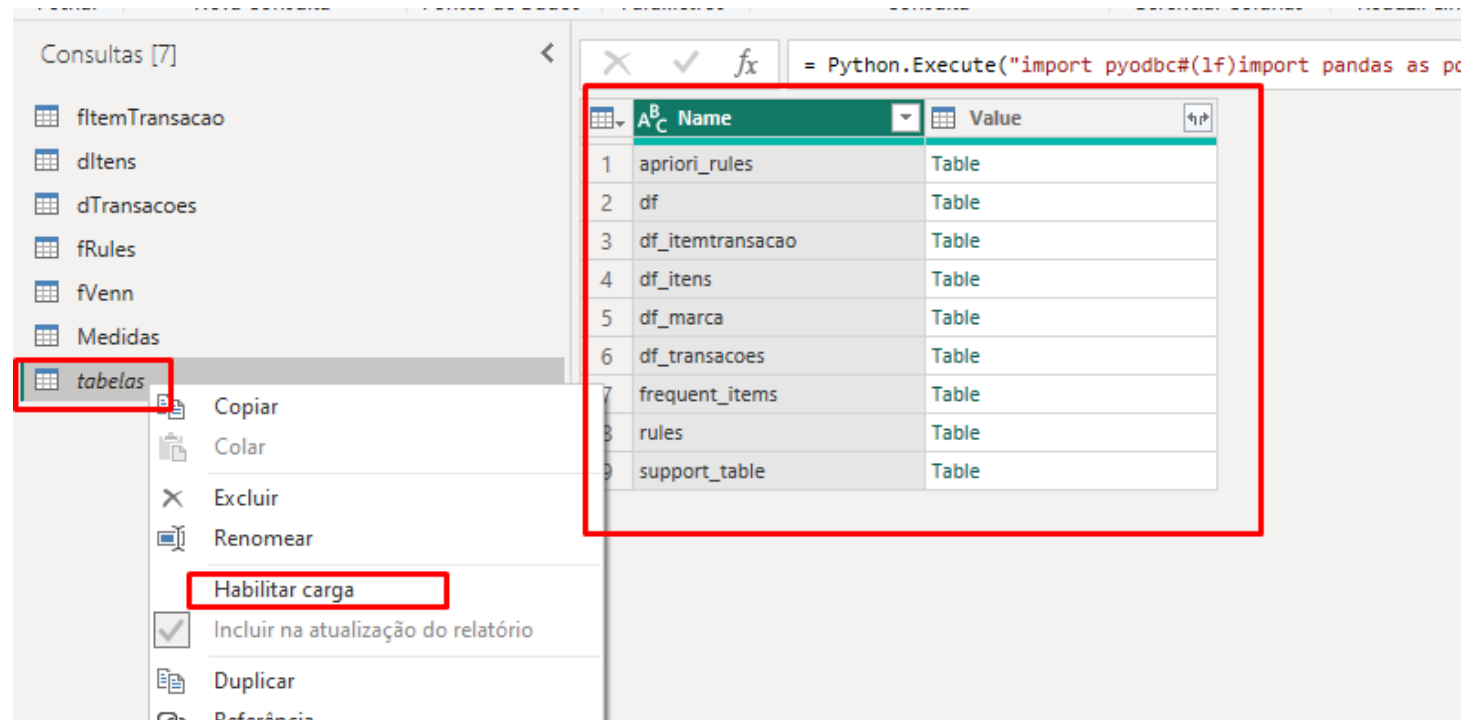
☒ Adicionar colunas de relação

OK

Cancelar

Modelagem Power BI

- Optou-se por manter apenas uma importação em uma tabela com a carga desabilitada, dessa forma consegue-se ganhar em *performance*, apenas referenciando a tabela "tabelas" nas demais.



Consultas [7]

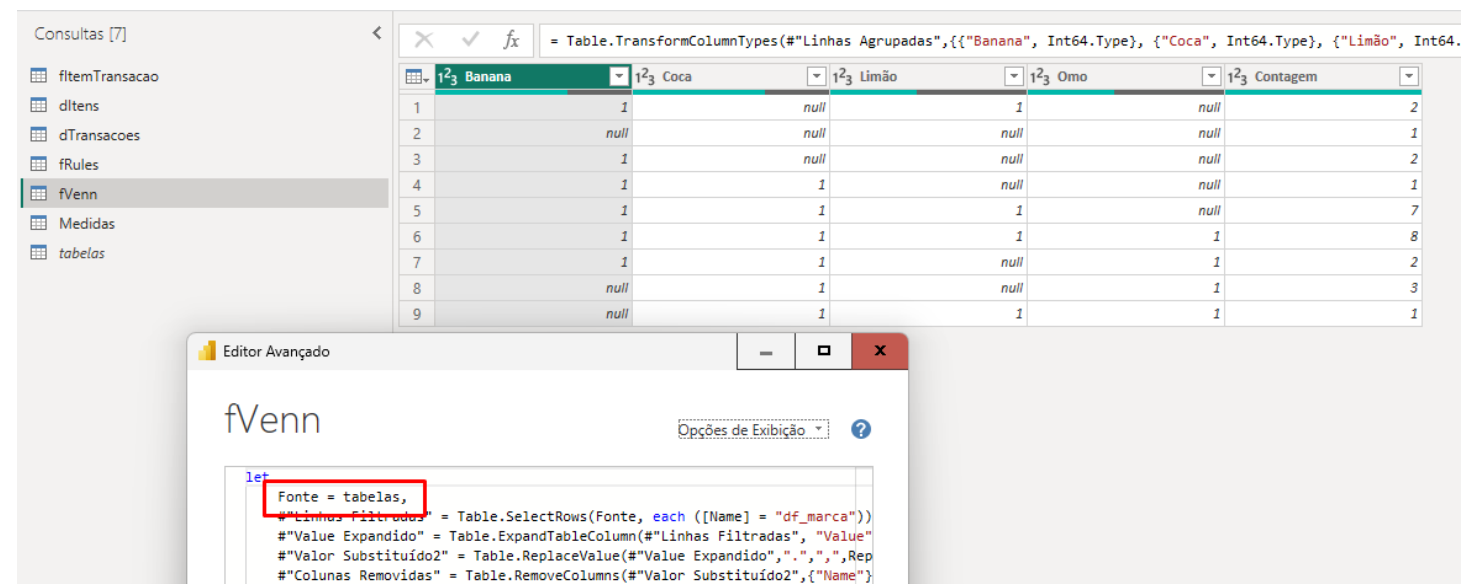
- fitemTransacao
- dltens
- dTransacoes
- fRules
- fVenn
- Medidas
- tabelas**

Context Menu:

- Copiar
- Colar
- Excluir
- Renomear
- Habilitar carga**
- ☒ Incluir na atualização do relatório
- Duplicar
- Referência

A ^B C	Name	Value
1	apriori_rules	Table
2	df	Table
3	df_itemtransacao	Table
4	df_itens	Table
5	df_marca	Table
6	df_transacoes	Table
7	frequent_items	Table
8	rules	Table
9	support_table	Table

Formula Bar: = Python.Execute("import pyodbc#(lf)import pandas as pc



Consultas [7]

- fitemTransacao
- dltens
- dTransacoes
- fRules
- fVenn**
- Medidas
- tabelas

1 ² 3	Banana	1 ² 3	Coca	1 ² 3	Limão	1 ² 3	Omo	1 ² 3	Contagem
1	1		1		1		1		2
2		1		1		1		1	1
3		1		1		1		1	2
4		1		1		1		1	1
5		1		1		1		1	7
6		1		1		1		1	8
7		1		1		1		1	2
8		1		1		1		1	3
9		1		1		1		1	1

Editor Avançado

fVenn

Opções de Exibição

```
let
    Fonte = tabelas,
    #Linhas Filtradas = Table.SelectRows(Fonte, each ([Name] = "df_marca")),
    #Value Expandido = Table.ExpandTableColumn(#Linhas Filtradas, "Value",
    #Valor Substituído2 = Table.ReplaceValue(#Value Expandido, ".", "", Rep
    #Colunas Removidas = Table.RemoveColumns(#Valor Substituído2, {"Name"}
```

Modelagem Power BI

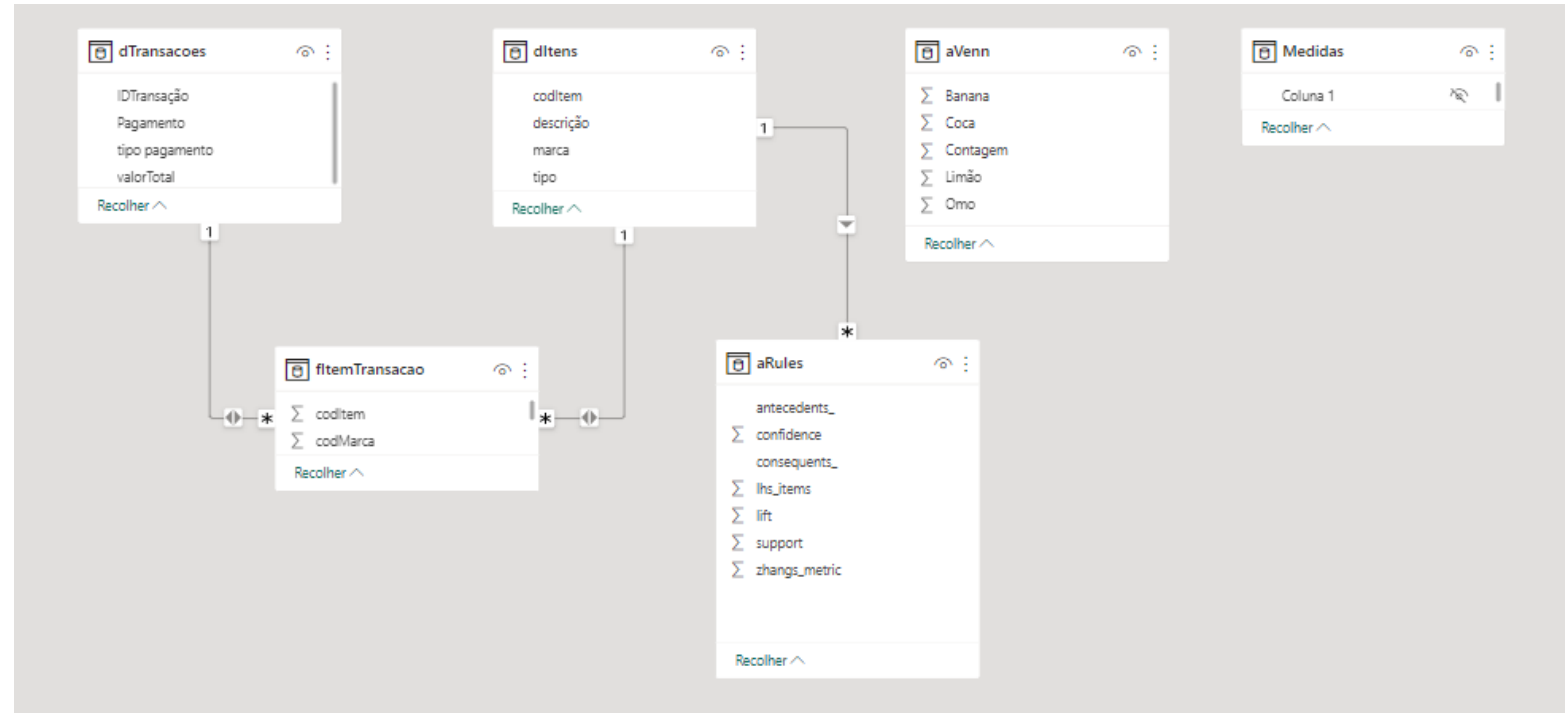
- Ainda na etapa de modelagem, foi inclusa uma coluna nas transações para facilitar a filtragem no visual pelo tipo de transação.

Nome do campo	Tipo de dados	Descrição (Opcional)
IDTransação	Numeração Automática	identificador de transação de compras
valorTotal	Número	valor total da compra
tipo pagamento	Texto Curto	indicação de pagamento em espécie (ES), cheque (CH) ou cartão de crédito (CC)

<pre>= Table.AddColumn(#"Tipo Alterado", "Pagamento", each if [tipo pagamento] = "es" then "Em Espécie" else if [tipo pagamento] = "ch" then "Cheque" else if [tipo pagamento] = "cc" then "Cartão de Crédito" else null)</pre>				
IDTransação	valorTotal	tipo pagamento	Pagamento	
1	15	ch	Cheque	
2	20	ch	Cheque	
3	14	es	Em Espécie	

Schema

- Foi gerado as conexões entre as tabelas com base no Star Schema, onde as tabelas fato interagem com as dimensões.
- Por padrão, a tabela "dTransacoes" seria selecionada como uma tabela de fatos devido a conter informações sobre as transações da operação. No entanto, devido à estrutura dos dados, ela foi tratada como uma entidade e foi assumido que "fItemTransacao" seria uma tabela de ligação. Entre a tabela de ligação e as dimensões, as conexões são de 1 para *, filtrando ambas as tabelas.



Resultado das Análises

- Como resultado foi identificado as associações com maior força, através de um ranqueamento.
- Pessoas que compram Limão Taiti e Coca Light, tem maior probabilidade de comprar também Omo Progress;
- E pessoas que compram Banana Caturra e Coca Light, tem maior probabilidade de comprar também Limão Siciliano.

```
1 Nota =
2 SUMX(
3     aRules,
4     [aRules[lift]+aRules[confidence]+aRules[support]]
5 )
```

```
1 Rank notas =
2 RANKX(
3     all(aRules),
4     [Nota]
5 )
```



Obrigada!

Rafaela Pacheco de Oliveira
Rafaelapo1997@gmail.com