IPOG

Mineração de Dados

EDITORA IPOG

IP5p Instituto de Pós-Graduação e Graduação – IPOG

Mineração de Dados. / Autor: Thiago Santana Lemes.

240f. :il.

ISBN:

1. Mineração de Dados 2. Ciência de Dados 3. Estatística 4. Matemática 5. Análise de Dados.

CDU: 005

Bibliotecário responsável: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

IPOG

Instituto de Pós Graduação e Graduação

http://www.ipog.edu.br

Sede

Av. T-1 esquina com Av. T-55 N. 2.390 - Setor Bueno - Goiânia-GO. Telefone (0xx62) 3945-5050

SUMÁRIO

APRESENTAÇÃO

Seja bem-vindo ou bem-vinda à disciplina de Mineração de Dados.

Essa disciplina é fundamental dentro do currículo da graduação em Ciência de Dados e em cursos de tecnologia. Este curso é projetado para equipar os estudantes com habilidades práticas e teóricas necessárias para extrair conhecimento útil de grandes volumes de dados. Ao longo do ciclo, os alunos aprenderão a aplicar técnicas de análise de dados para resolver problemas do mundo real.

Os alunos irão explorar os conceitos básicos e as metodologias empregadas na mineração de dados, incluindo a preparação de dados, análise estatística por meio de indicadores e gráficos, redução de dimensionalidade, e mineração de padrões frequentes. Através de projetos práticos, os estudantes aplicarão ferramentas e técnicas para analisar conjuntos de dados reais, praticando a construção de modelos de mineração de dados eficazes.

A mineração de dados é aplicável em uma variedade de campos, oferecendo soluções para problemas complexos em setores como marketing, onde pode ajudar a entender melhor o comportamento do consumidor; saúde, facilitando diagnósticos mais precisos e tratamentos personalizados; finanças, otimizando estratégias de investimento e gestão de riscos; e muito mais. O curso é estruturado para que o aluno possa aprender a aplicar essas técnicas em contextos reais, preparando-os para enfrentar desafios contemporâneos em qualquer área que dados sejam um recurso crucial.

Este curso é ideal para aqueles que desejam aprofundar seus conhecimentos em técnicas de análise de dados e aspiram a posições que requerem habilidades avançadas em análise de dados, pesquisa operacional ou inteligência de negócios. Preparem-se para explorar o vasto e excitante campo da mineração de dados e para aplicar o conhecimento adquirido de forma criativa e impactante em diversas áreas do conhecimento.

Estou entusiasmado para ajudá-los a descobrir o poder da mineração de dados e a impactar positivamente o mundo com as habilidades aprendidas aqui. Prepare-se para uma jornada desafiadora e gratificante!

Boa leitura e estudos!

Prof. Dr. Thiago Santana Lemes

OBJETIVOS

OBJETIVO GERAL

Proporcionar uma compreensão dos princípios, algoritmos e aplicações da mineração de dados, capacitando os estudantes a aplicar esses conhecimentos na solução de problemas reais, na análise de dados e no desenvolvimento de sistemas inteligentes.

OBJETIVOS ESPECÍFICOS

- Conhecer os conceitos fundamentais da mineração de dados incluindo redução de dimensionalidade, agrupamento e mineração de padrões frequentes;
- Desenvolver habilidades práticas em modelagem, seleção de algoritmos, ajuste de modelos e avaliação de desempenho;
- Entender os fundamentos da análise exploratória de dados;
- Compreender questões éticas e o impacto social da mineração de dados, incluindo viés algorítmico, privacidade de dados, e responsabilidade na tomada de decisões.

Conheça como esse conteúdo foi organizado!

Unidade 1: Conceitos iniciais de data mining

Unidade 2: Análise exploratória dos dados

Unidade 3: Redução de dimensionalidade

Unidade 4: Clusterização

Unidade 5: Mineração de padrões frequentes

UNIDADE 1 CONCEITOS INICIAIS DE MINERAÇÃO DE DADOS

É com grande prazer que damos início a essa jornada de aprendizado e descobertas. Nesta unidade, teremos a oportunidade de explorar os conceitos iniciais do fascinante universo de mineração de dados.

OBJETIVOS DA UNIDADE 1

Ao final dos estudos, você deverá ser capaz de:

- Definir e utilizar conceitos básicos de mineração de dados.
- Pré-processar conjuntos de dados
- Configurar o ambiente de desenvolvimento.

1.1 INTRODUÇÃO À MINERAÇÃO DE DADOS

A mineração de dados é uma área antiga e que evoluiu bastante ao longo dos anos. A crescente evolução da tecnologia trouxe consigo computadores mais poderosos, diversos recursos de armazenamento de dados em nuvem em grandes quantidades (*big data*) e uma aquisição de dados cada vez mais acelerada. Para fazer uso de tantos dados e conseguir obter vantagens estratégicas é necessário lançar mão de inúmeras técnicas de análise de dados e ferramentas capazes de processar essas informações. No intuito de suportar as necessidades de entendimento desses conjuntos de dados é que surge a Descoberta de Conhecimento em Bases de Dados (*Knowledge Discovery in Databases* – KDD).

O termo KDD foi formalizado por um grupo de pesquisadores no fim dos anos 80 e pode ser entendido como um processo de várias etapas que é complexo e envolve a atuação de profissionais com experiência, sendo possível existir procedimentos repetitivos no intuito de encontrar padrões em grandes bases de dados (Goldschmidt & Passos, 2005). Além disso, o KDD é um processo que tem como base diversas etapas operacionais, sendo as principais: Pré-processamento, Mineração dos Dados e Pós-processamento (Goldschmidt & Passos, 2005).

A etapa de pré-processamento envolve basicamente organizar, limpar e transformar os dados deixando-os prontos para serem utilizados nos algoritmos. A etapa de mineração de dados tem como objetivo extrair informações dos dados e tornar evidente padrões que possam ser utilizados. É nessa etapa que o conhecimento se torna evidente e fica disponível para ser utilizado. A etapa de pós-processamento nem sempre é aplicada, porém funciona como uma consolidação do conhecimento obtido.

Vale reforçar então que a mineração de dados é uma etapa dentro de KDD e que ambos os conceitos são extremamente importantes em diversas áreas atualmente como saúde, *marketing*, indústria, varejo, educação dentre outras. O conhecimento a respeito de padrões, tendências e comportamentos em geral é muito valioso e promove uma vantagem competitiva muito grande no mercado. A digitalização de grandes empresas aliado a todos os investimentos em tecnologia, dados e inteligência artificial tornou possível grandes transformações na sociedade. O estudo da área de dados nunca foi tão essencial quanto hoje e vai continuar crescendo ao longo dos próximos anos.

1.2 AMBIENTE DE DESENVOLVIMENTO

O Google fornece uma plataforma de desenvolvimento para projetos de ciência de dados e inteligência artificial chamada Colab. Esse ambiente é gratuito para utilização, podendo fornecer recursos computacionais extras por meio de pagamento. A Figura 1.1 abaixo mostra a página inicial que pode ser acessada por meio do link https://colab.research.google.com.

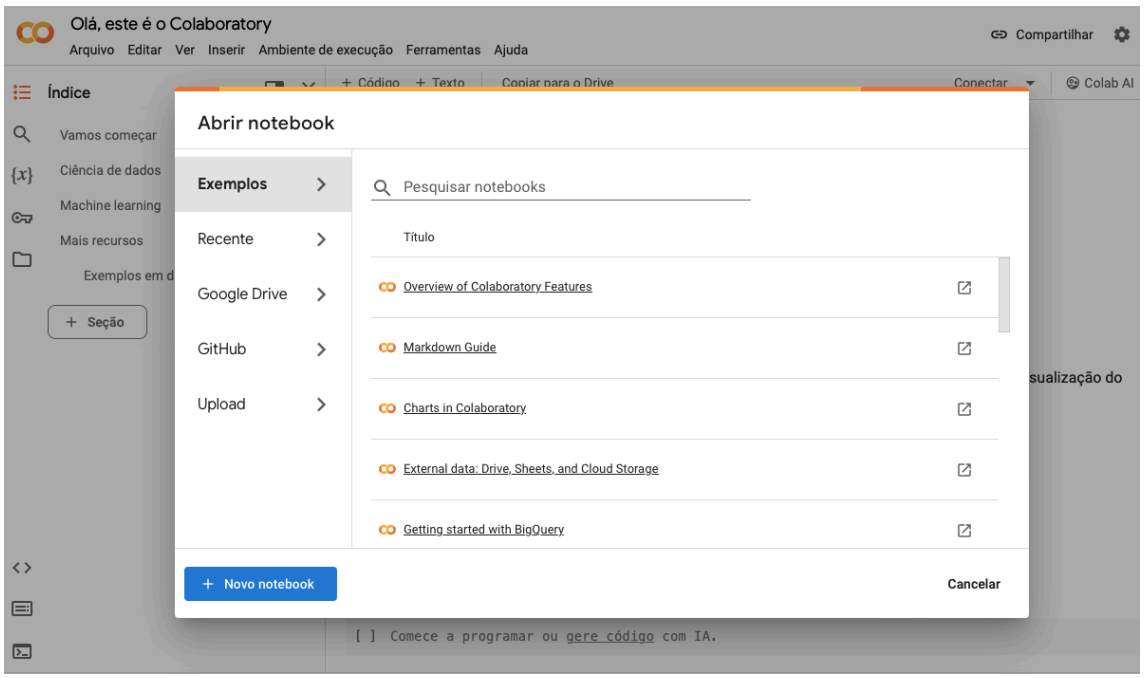

Figura 1.1: Página inicial do Google Colab

A plataforma possui diversos exemplos e tutoriais e todos estão acessíveis já na aba Exemplos como mostrado na Figura 1.1. Para iniciar um projeto basta clicar no botão azul Novo notebook. Todos os códigos que serão apresentados nesse texto podem ser desenvolvidos no ambiente do Google Colab. A vantagem é que a maioria das bibliotecas padrão no universo de *data science* e *machine learning* já estão instaladas facilitando muito o início do desenvolvimento de um projeto. Obviamente é possível instalar bibliotecas caso elas não estejam disponíveis de início no ambiente. Para fazer isso basta utilizar um comando no formato: *!pip install "nome_da_biblioteca"*.

1.3 LIMPEZA DE DADOS

Na maioria dos projetos reais que envolvem análise de dados existem problemas relacionados à qualidade dos dados. Esses problemas podem surgir por diversos motivos, como por exemplo um conjunto de dados que é formado pela aquisição de informações de sensores e que em alguns momentos falham, deixando registros em branco, ou mesmo dados provenientes de pesquisas, os quais podem ter registros faltantes. Para lidar com esses dados incompletos existem diversas abordagens, entre elas, preencher os elementos faltantes com a média ou a mediana dos registros ou então remover os registros da análise.

A remoção tem a desvantagem da perda de informações, mas em alguns casos não há solução, por outro lado, o preenchimento com a média insere valores que não são exatos e para cada projeto é necessário validar qual a melhor estratégia. A Figura 1.2 abaixo ilustra um conjunto de dados implementado por meio da biblioteca Pandas (pandas *dataframe*) (The pandas development team, 2024) com alguns registros faltantes (NaN – *Not a Number*)

	Sexo	Escolaridade	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	M	Superior	NaN	1	1.67	78.0	True
1	NaN	Médio	Sim	1	1.80	NaN	False
2	F	Médio	Não	1	1.88	101.0	NaN
3	M	NaN	Sim	1	NaN	77.0	False
4	M	Médio	Não	1	1.73	65.0	True
5	M	Pós-graduação	Não	0	1.72	90.0	False
6	F	Médio	Não	1	1.80	97.0	False
7	F	Médio	Sim	0	1.77	NaN	NaN

Figura 1.2: *Dataset* com valores faltantes

Uma maneira de preencher os dados numéricos com a média é utilizar o módulo *SimpleImputer* da biblioteca *Scikit-learn* (Pedregosa et al., 2011). O trecho de código apresentado na Figura 1.3 abaixo ilustra a implementação dessa abordagem. Na sequência, a Figura 1.4 mostra o conjunto de dados sem valores nulos nas colunas Altura e Peso.

```
1 from sklearn.impute import SimpleImputer
2 imputer = SimpleImputer(strategy='mean')
3 matriz_preenchida = imputer.fit_transform(df[['Altura', 'Peso']])
4 df['Altura'] = np.round(matriz_preenchida[:, 0],2)
5 df['Peso'] = np.round(matriz_preenchida[:, 1],2)
6 df
```

Figura 1.3: Implementação do *SimpleImputer* para preenchimento de dados com a média

	Sexo	Escolaridade	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	M	Superior	NaN	1	1.67	78.00	True
1	NaN	Médio	Sim	1	1.80	84.67	False
2	F	Médio	Não	1	1.88	101.00	NaN
3	M	NaN	Sim	1	1.77	77.00	False
4	M	Médio	Não	1	1.73	65.00	True
5	M	Pós-graduação	Não	0	1.72	90.00	False
6	F	Médio	Não	1	1.80	97.00	False
7	F	Médio	Sim	0	1.77	84.67	NaN

Figura 1.4: *Dataframe* após aplicação do *SimpleImputer* para preencher dados numéricos de Altura e Peso com a média

Para remover todos os registros nulos de uma só vez é possível utilizar o comando *dropna()* da biblioteca Pandas. Veja o trecho de código apresentado na Figura 1.5 abaixo.

```
df.dropna(inplace=True, ignore_index=True)
df
```

Figura 1.5: Remoção de dados nulos em um *dataframe*

Na Figura 1.5 acima, *inplace* é um parâmetro que indica que a operação de remoção deve ser executada permanentemente na memória. Além disso, *ignore_index* faz com que os índices do *dataframe* sejam numerados novamente de 0 até o fim. O *dataframe* é apresentado na Figura 1.6 abaixo.

	Sexo	Escolaridade	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	M	Médio	Não	1	1.73	65.0	True
1	M	Pós-graduação	Não	0	1.72	90.0	False
2	F	Médio	Não	1	1.80	97.0	False

Figura 1.6: Conjunto de dados após remoção de dados nulos

No caso da remoção é perceptível a perda de registros que é sempre ruim, mas em alguns casos é a única maneira devido à qualidade dos dados. Existe ainda uma outra possibilidade nativa da biblioteca Pandas que é o *fillna()*, na

qual todos os valores nulos são preenchidos com o valor informado entre os parêntesis. Fica claro que existem maneiras distintas de tratar dados nulos e a melhor forma deve ser sempre avaliada com cautela.

1.4 CONVERSÃO DE DADOS

Outra questão importante em projetos de mineração de dados é a conversão de dados categóricos em numéricos. Isso se deve ao fato de que a maioria dos modelos só conseguem processar dados numéricos. Para clarificar esse conceito vamos tomar como exemplo o *dataset* da Figura 1.7 abaixo que é uma versão sem valores nulos da que foi apresentada na Figura 1.2. Temos a variável (coluna) “Sexo” com valores “M” ou “F”. Nesse caso, “M” pode ser representado por 1 e “F” por 0.

	Sexo	Escolaridade	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	M	Superior	Sim	1	1.67	78.0	True
1	F	Médio	Sim	1	1.80	83.0	False
2	F	Médio	Não	1	1.88	101.0	False
3	M	Superior	Sim	1	1.71	77.0	False
4	M	Médio	Não	1	1.73	65.0	True
5	M	Pós-graduação	Não	0	1.72	90.0	False
6	F	Médio	Não	1	1.80	97.0	False
7	F	Médio	Sim	0	1.77	NaN	True

Figura 1.7: *Dataset* sem dados nulos

A variável “Escolaridade” apresenta uma situação interessante pois possui três valores distintos (Superior, Médio e Pós-graduação). A mesma lógica empregada na variável “Sexo” pode ser aplicada aqui, mas pela quantidade maior de opções irá gerar mais colunas. Além disso, a variável “Possui casa própria” também pode ser modificada seguindo a lógica da variável “Sexo” e utilizar 1 ou 0, porém agora no lugar de “Sim” e “Não”.

	Sexo	Escolaridade_Pós-graduação	Escolaridade_Superior	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	1	0	1	1	1	1.67	78.0	True
1	0	0	0	1	1	1.80	83.0	False
2	0	0	0	0	1	1.88	101.0	False
3	1	0	1	1	1	1.71	77.0	False
4	1	0	0	0	1	1.73	65.0	True
5	1	1	0	0	0	1.72	90.0	False
6	0	0	0	0	1	1.80	97.0	False
7	0	0	0	1	0	1.77	NaN	True

Figura 1.8: *Dataset* com variáveis categóricas convertidas

A Figura 1.8 acima apresenta o *dataset* contendo a nova configuração em que as variáveis categóricas foram convertidas. Vale destacar que a variável “Escolaridade” agora precisa de duas colunas para ser representada: “Escolaridade_Pós-graduação” e “Escolaridade_Superior”. Quando a amostra se referir a uma pessoa com Pós-graduação irá aparecer 1 na coluna “Escolaridade_Pós-graduação” e 0 na coluna “Escolaridade_Superior”. No caso em que ambas as colunas forem 0 automaticamente indica que a escolaridade é “Médio”. O trecho de código abaixo (Figura 1.9) mostra a implementação completa para fazer a conversão das variáveis categóricas.

```
df_categorico = pd.get_dummies(df[['Sexo', 'Escolaridade', 'Possui casa própria']], drop_first=True, dtype=int)
df_categorico.rename(columns={'Sexo_M':'Sexo', 'Possui casa própria_Sim':'Possui casa própria'}, inplace=True)
df.drop(['Sexo', 'Escolaridade', 'Possui casa própria'], axis=1, inplace=True)
df = pd.concat([df_categorico, df], axis=1)
```

Figura 1.9: Implementação da conversão de dados categóricos utilizando a biblioteca Pandas

Esse tipo de abordagem na qual variáveis categóricas são convertidas em numéricas é também conhecido como *One-Hot-Encoding*. As variáveis binárias criadas para representar as variáveis categóricas são conhecidas como *dummy*.

1.5 ENRIQUECIMENTO DE DADOS

O enriquecimento de dados pode ser visto como uma melhoria do conjunto de dados atuais. Essa abordagem pode envolver transformações e padronizações dos dados, adição de novos dados provenientes de outras fontes que não as iniciais, limpeza dos dados e criação de novos dados com base nos originais. Nesse texto, iremos dar foco na possibilidade de criação de uma nova coluna com base em outras colunas já existentes. Vamos utilizar o conjunto de dados apresentado na Figura 1.8 e criar uma nova coluna com base na “Altura” e “Peso”. Utilizando essas duas características é possível calcular o Índice de Massa Corporal (IMC) que é dado pela expressão abaixo:

$$imc = peso * altura^2 \qquad \text{Equação 1.1}$$

Para calcular o IMC e criar uma nova coluna contendo seu valor podemos utilizar o método *apply()* do Pandas e aplicar ao *dataframe* como mostra o trecho de código abaixo.

```
df['imc'] = df[['Altura', 'Peso']].apply(lambda x : np.round(x['Peso']/x['Altura']**2,2), axis=1)
```

A Figura 1.10 abaixo mostra o *dataframe* contendo a nova coluna IMC.

	Sexo	Escolaridade_Pós-graduação	Escolaridade_Superior	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico	imc
0	1	0	1	1	1	1.67	78.0	True	27.97
1	0	0	0	1	1	1.80	83.0	False	25.62
2	0	0	0	0	1	1.88	101.0	False	28.58
3	1	0	1	1	1	1.71	77.0	False	26.33
4	1	0	0	0	1	1.73	65.0	True	21.72
5	1	1	0	0	0	1.72	90.0	False	30.42
6	0	0	0	0	1	1.80	97.0	False	29.94
7	0	0	0	1	0	1.77	68.0	True	21.71

Figura 1.10: *Dataframe* contendo a coluna IMC

O exemplo acima de criação de uma nova coluna é bastante comum na área de dados e trás novas perspectivas. Devemos nos atentar para o contexto do problema, visto que nem sempre é possível criar novas colunas.

1.6 TRANSFORMAÇÃO DE DADOS

Outro ponto crucial na etapa de pré-processamento dos dados é o de transformação, o qual leva todos os valores registrados para um mesmo patamar de comparação. O *dataset* da Figura 1.10 nos ajuda a entender essa situação ao tomar como exemplo as variáveis “Altura”, “Peso” e “imc”. De forma geral, a altura de uma pessoa é um número maior do que zero e menor do que 2,5, considerando a unidade de medida metros (m). O peso de uma pessoa também é um número maior do que zero e normalmente abaixo de 150, considerando a unidade de medida quilogramas (kg). Vale ressaltar que obviamente podem existir pessoas com medidas superiores às comentadas, mas aqui nesse texto, iremos nos ater a poucos valores para fins pedagógicos. Considerando essa explicação, vemos que os valores de peso e altura estão em patamares bem diferentes, e isso pode atrapalhar em alguns casos a descoberta de padrões ou mesmo levar modelos de *Machine Learning* a não compreenderem bem a relação entre os dados, ou mesmo atribuir maior importância a um conjunto por possuir valores muito maiores do que o outro conjunto. Para mitigar esse problema, aplica-se um procedimento de transformação aos dados que padroniza todos baseado em uma lógica. A Figura 1.11 abaixo mostra o mesmo *dataset* da Figura 1.10, mas agora com as variáveis “Altura”, “Peso” e “imc” transformadas.

	Sexo	Escolaridade_Pós-graduação	Escolaridade_Superior	Possui casa própria	Possui trabalho CLT	Já realizou algum procedimento cirúrgico	Altura	Peso	imc
0	1	0	1	1	1	True	0.0000	0.3611	0.7187
1	0	0	0	1	1	False	0.6190	0.5000	0.4489
2	0	0	0	0	1	False	1.0000	1.0000	0.7887
3	1	0	1	1	1	False	0.1905	0.3333	0.5304
4	1	0	0	0	1	True	0.2857	0.0000	0.0011
5	1	1	0	0	0	False	0.2381	0.6944	1.0000
6	0	0	0	0	1	False	0.6190	0.8889	0.9449
7	0	0	0	1	0	True	0.4762	0.0833	0.0000

Figura 1.11: *Dataset* com variáveis “Altura”, “Peso” e “imc” transformadas

Vale destacar que existem vários tipos de métodos para transformar os dados, dentre eles se destacam o mínimo e máximo (*MinMax*) e a estandardização (*Standardization*). Em muitos casos, uma certa transformação pode ajudar os algoritmos a performarem melhor e por isso, cabe ao profissional de Ciência de Dados juntamente com a equipe do projeto definir qual abordagem é a mais adequada. A Figura 1.11 mostra o resultado da transformação após a utilização do método *MinMax* da biblioteca *Scikit-learn* que tem por base obter o valor mínimo e o valor máximo no conjunto observado. A coluna “Peso” possuía o valor máximo de 101 na terceira linha e após o *MinMax* se transformou em 1, enquanto o valor mínimo era de 77 e agora é 0. Dessa forma, a lógica aqui é transformar todos os valores que estão entre 77 e 101 para uma nova faixa de valores correspondentes entre 0 e 1. O trecho de código apresentado na Figura 1.12 abaixo mostra a implementação da transformação dos dados.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
dados_min_max = scaler.fit_transform(df[['Altura', 'Peso', 'imc']])
df_min_max = pd.DataFrame(data=np.round(dados_min_max,4), columns=['Altura', 'Peso', 'imc'])
df.drop(['Altura', 'Peso', 'imc'], axis=1, inplace=True)
df = pd.concat([df, df_min_max], axis=1)
```

Figura 1.12: Implementação da transformação de dados utilizando *MinMax*

UNIDADE 2 ANÁLISE EXPLORATÓRIA DE DADOS

Tivemos uma ideia básica a respeito da mineração de dados fazendo algumas manipulações, transformações e conversões de dados. Foi possível perceber que existem diversos conceitos envolvidos e que eles são essenciais para o bom andamento de um projeto que tem por base os dados.

Nessa nova unidade iremos caminhar no sentido de explorar de forma básica os dados, aplicando conceitos estatísticos, gerando gráficos para visualizar os dados e tentando encontrar relações interessantes.

OBJETIVOS DA UNIDADE 2

Ao final dos estudos, você deverá ser capaz de:

- Realizar a análise exploratória dos dados
- Descobrir relações básicas entre os dados

2.1. MEDIDAS DE TENDÊNCIA CENTRAL

Sempre que se inicia um projeto de mineração de dados uma análise exploratória de dados é necessária. As tomadas de decisão, escolha de algoritmos e escolha de apresentação de resultados passa pelo completo entendimento do conjunto de dados que está sendo utilizado.

Como primeiros passos, é preciso visualizar todas as variáveis disponíveis, entender os seus tipos e como estão distribuídos. Variáveis numéricas podem ser provenientes de medições ou registros como altura e peso de indivíduos ou mesmo indicações de que algo estava ligado ou desligado (funcionando ou não funcionando, presente ou ausente, ...) ou se determinada ação foi tomada ou não. Variáveis que indicam dois estados são conhecidas como binárias e podem ser representadas por palavras (masculino/feminino - *string*) ou por booleanos (Verdadeiro/Falso – *True/False*). Além disso, essa lógica de representar dois estados pode ser estendida para diversos outros. A Figura 2.1 a seguir ilustra os tipos de variáveis do *dataframe* já apresentado na Figura 1.7 e que foi explorado na seção anterior. Para gerar os tipos de variáveis foi utilizado o método *info()* da biblioteca Pandas.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 7 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Sexo                                    8 non-null      object
 1   Escolaridade                            8 non-null      object
 2   Possui casa própria                     8 non-null      object
 3   Possui trabalho CLT                     8 non-null      int64
 4   Altura                                  8 non-null      float64
 5   Peso                                    8 non-null      float64
 6   Já realizou algum procedimento cirúrgico  8 non-null      bool
dtypes: bool(1), float64(2), int64(1), object(3)
memory usage: 520.0+ bytes
```

Figura 2.1: Tipos de variáveis no *dataframe*

Na Figura 2.1, as variáveis “Sexo”, “Escolaridade” e “Possui casa própria” são do tipo texto (*string*). O Pandas apresenta essas variáveis do tipo texto como *object*. “Possui trabalho CLT” é do tipo inteiro e o Pandas apresenta como *int64*. As variáveis “Altura” e “Peso” são do tipo *float* e o Pandas apresenta *float64*. “Já realizou algum procedimento cirúrgico” é uma variável do tipo *bool*, visto que só apresenta valores *True* ou *False*. O comando *info()* do Pandas é extremamente útil pois consolida de maneira simples todas as variáveis que fazem parte do *dataframe*, facilitando a visualização e conferência de informações.

Após essa inspeção inicial nos dados é muito importante verificar as medidas de tendência central, dispersão dos dados, correlações entre variáveis e finalmente uma representação visual. A Figura 2.2 abaixo mostra a utilização do comando *describe()* do Pandas que retorna de uma só vez a média, o desvio padrão, valores mínimos e máximos e os percentis dos dados, sendo um comando extremamente útil.

```
df[['Altura', 'Peso']].describe()
```

	Altura	Peso
count	8.000000	8.000000
mean	1.760000	82.375000
std	0.066332	12.960793
min	1.670000	65.000000
25%	1.717500	74.750000
50%	1.750000	80.500000
75%	1.800000	91.750000
max	1.880000	101.000000

Figura 2.2: Descrição dos dados utilizando Pandas e *describe*

Para calcular os valores de média, mediana e desvio padrão separadamente é possível utilizar respectivamente os comandos *mean()*, *median()*, *std().*

```python
print(f"Média: {np.round(df['Altura'].mean(),4)}")
print(f"Mediana: {np.round(df['Altura'].median(),4)}")
print(f"Desvio padrão: {np.round(df['Altura'].std(),4)}")
```

```
Média: 1.76
Mediana: 1.75
Desvio padrão: 0.0663
```

```python
print(f"Média: {np.round(df['Peso'].mean(),4)}")
print(f"Mediana: {np.round(df['Peso'].median(),4)}")
print(f"Desvio padrão: {np.round(df['Peso'].std(),4)}")
```

```
Média: 82.375
Mediana: 80.5
Desvio padrão: 12.9608
```

Figura 2.3: Cálculo da média, mediana e desvio padrão separados por coluna

A Figura 2.3 acima ilustra a implementação do cálculo das medidas de tendência central em cada coluna separadamente. Além disso, o comando *round()* da biblioteca *Numpy* (Harris et al., 2020) foi utilizado para arredondar a resposta exibida pelo *print()* em quatro casas decimais. Vale ressaltar que é possível calcular todas essas medidas de uma só vez para todas as colunas numéricas, mas aqui, para fins de ilustração com *print*, foi feito de forma separada. O cálculo da moda (valor que mais ocorre em um conjunto de dados) pode ser feito por meio do comando *mode()* ou mesmo por meio do comando *value_counts()*. Especialmente o comando *value_counts()* apresenta a quantidade de valores únicos. O elemento que possui mais valores iguais é a moda. A Figura 2.4 abaixo mostra esse comando aplicado à coluna “Altura” e o valor de 1,80 aparecendo no topo com 2 repetições no conjunto de dados.

```
1 df['Altura'].value_counts()
```

```
Altura
1.80    2
1.67    1
1.88    1
1.71    1
1.73    1
1.72    1
1.77    1
Name: count, dtype: int64
```

Figura 2.4: Quantidade de valores únicos

2.2. CORRELAÇÃO

A correlação é uma medida estatística que avalia a relação entre duas variáveis. Para verificar a correlação entre as variáveis pode-se utilizar o comando *corr()* da biblioteca Pandas como mostrado na Figura 2.5 abaixo.

```
df[['Altura', 'Peso']].corr()
```

	Altura	Peso
Altura	1.000000	0.594877
Peso	0.594877	1.000000

Figura 2.5: Correlação entre as variáveis

Esse comando devolve um *dataframe* contendo as correlações ao se analisar a linha e a coluna. A diagonal principal é a correlação perfeita (1.0) visto que é a comparação da variável com ela mesma. Valores mais próximos de 1 indicam uma correlação forte no sentido positivo, ou seja, as variáveis aumentam juntas. Valores mais próximos de -1 indicam uma correlação forte no sentido negativo, ou seja, enquanto uma variável aumenta a outra diminui. As variáveis nesse exemplo, “Peso” e “Altura” possuem uma correlação de 0.59 aproximadamente, ou 59% que não indica uma correlação muito forte.

Uma outra maneira interessante de verificar a correlação entre variáveis é através de um mapa de calor. As bibliotecas *Matplotlib* e *Seaborn* (Hunter, 2007; Waskom, 2021) ajudam nessa tarefa. Veja a Figura 2.6 a seguir.

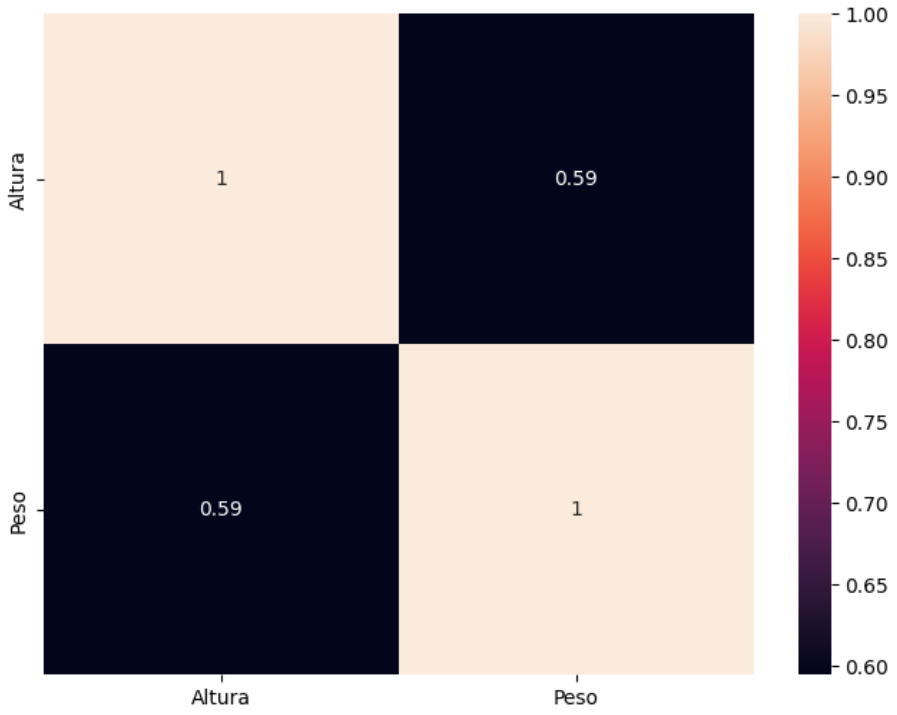

Figura 2.6: Mapa de calor das variáveis Altura e Peso

É possível perceber na Figura 2.6 que a cor mais clara representa a correlação 1.0. Além disso, de acordo com a escala de cor, quanto mais escuro, mais fraca a correlação se torna. Obviamente esse exemplo é muito simples e só ilustra a comparação entre duas variáveis, porém, todos os passos anteriores se aplicam para quantidades bem maiores de dados. Para gerar a Figura 2.6 foi utilizado o trecho de código abaixo na Figura 2.7:

```
1 fig, ax = plt.subplots(figsize=(8,6))
2 sns.heatmap(df[['Altura', 'Peso']].corr(), annot=True, ax=ax)
```

Figura 2.7: Implementação do código para gerar o mapa de calor

Além do mapa de calor é possível fazer um gráfico de dispersão (*scatter plot*) para verificar a relação entre duas variáveis. Veja a Figura 2.8 a seguir.

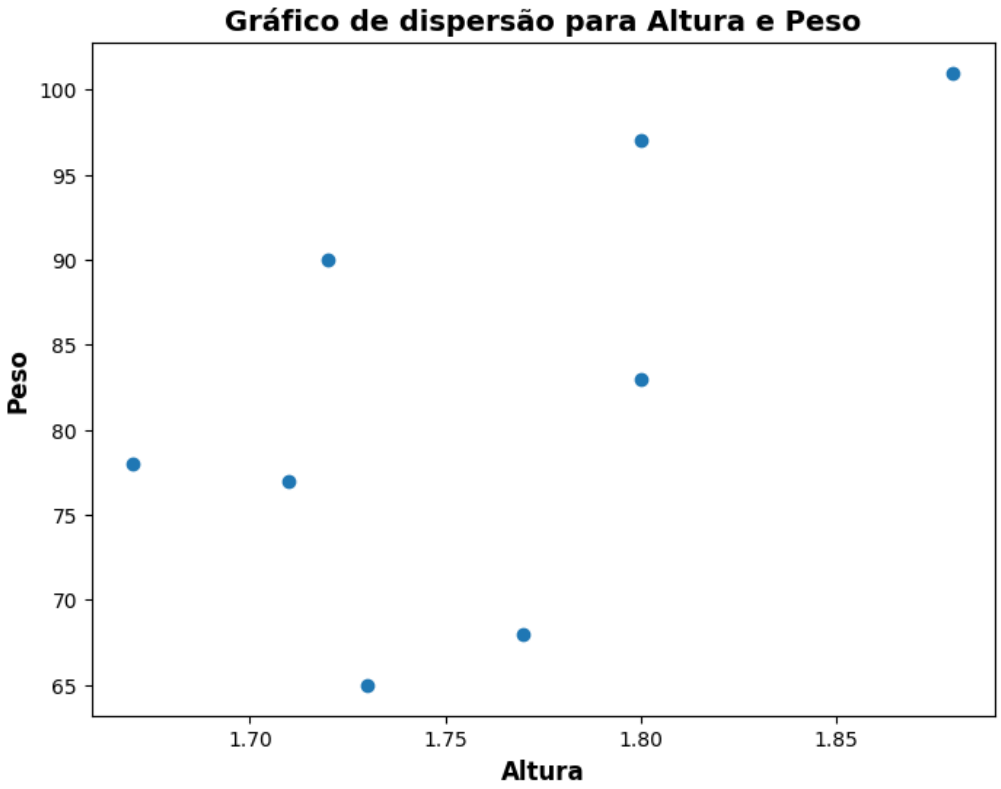

Figura 2.8: Gráfico de dispersão para as variáveis Altura e Peso

No gráfico acima é possível perceber uma certa tendência de aumento no peso à medida em que o valor da altura aumenta. A Figura 2.8 foi gerada por meio do trecho de código abaixo (Figura 2.9).

```
fig, ax = plt.subplots(figsize=(8,6))
ax.scatter(df['Altura'], df['Peso'])
ax.set_xlabel('Altura', fontsize=12, fontweight='bold')
ax.set_ylabel('Peso', fontsize=12, fontweight='bold')
ax.set_title('Gráfico de dispersão para Altura e Peso', fontsize=14, fontweight='bold')
```

Figura 2.9: Código para implementação do gráfico de dispersão

2.3. ANÁLISE DE OUTLIERS

Valores discrepantes, também conhecidos como *outliers* são comuns em diversos conjuntos de dados. Tais valores podem ocorrer devido à registro incorreto de informações, seja por eventos especiais em determinado registro, por exemplo, em um conjunto de dados de vendas de uma loja, uma promoção pode fazer com que as vendas aumentem muito gerando valores muito maiores do que o normal. Dessa forma, os *outliers* fogem ao padrão da distribuição comum dos dados. Os *outliers* portanto precisam ser investigados com calma e necessitam de um entendimento do contexto do problema. Em muitos casos, os outliers podem ser removidos, em outros, eles precisam ser mantidos, porém com a atenção a esse contexto especial. Veja a figura abaixo que ilustra dados a respeito de algumas cidades Brasileiras:

	Município	Estado	População
0	São Paulo	SP	11451245
1	Rio de Janeiro	RJ	6211423
2	Brasília	DF	2817068
3	Fortaleza	CE	2428678
4	Salvador	BA	2418005
5	Belo Horizonte	MG	2315560
6	Manaus	AM	2063547
7	Curitiba	PR	1773733
8	Recife	PE	1488920
9	Goiânia	GO	1437237

Figura 2.10: Dados de algumas cidades Brasileiras

É fácil perceber olhando os dados da figura acima que a cidade de São Paulo é um *outlier* em se tratando de população, visto que possui muito mais habitantes em relação às outras capitais. O Rio de Janeiro também possui muitos habitantes e está em segundo lugar em quantidade. Uma maneira interessante de visualizar os dados é por meio de um diagrama de caixa (*boxplot*). Veja a Figura 2.11 abaixo.

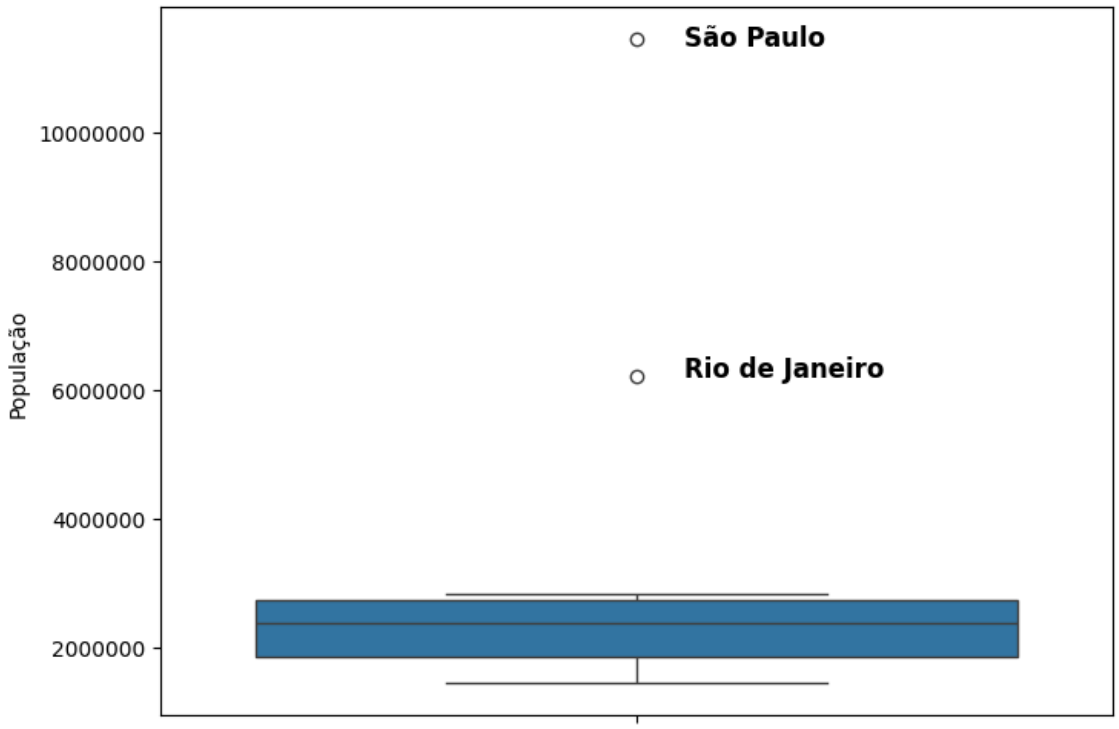

Figura 2.11: *Boxplot* referente às cidades Brasileiras

Em um diagrama de caixa a linha que fica aproximadamente no meio do retângulo é a mediana (50%) dos dados a borda superior representa o percentil de 75% e a linha inferior o percentil 25%. As duas linhas externas ao retângulo representam os limites superior e inferior e tudo que está acima ou abaixo dessas linhas é considerado um *outlier*. Na figura acima é fácil visualizar os pontos referentes à São Paulo e Rio de Janeiro muito acima do limite superior, sendo obviamente *outliers*.

Vamos agora analisar um exemplo de remoção de *outliers*. Vale destacar que o exemplo é meramente ilustrativo para clarificar o processo de remoção e não quer dizer que em um projeto real seja sempre necessário fazer isso, devendo-se validar o contexto dos dados e o objetivo final. Será utilizado um conjunto de dados referente ao problema de transações fraudulentas em *e-commerces* disponível em *link*. Esse conjunto de dados, originalmente possui mais colunas do que será apresentado nesse material, porém, os dados mantidos aqui são suficientes para o entendimento.

Veja na Figura 2.12 a seguir os primeiros cinco registros desse *dataset*, que possui no total 1.472.952 registros.

	Quantidade de transações	Método de Pagamento	Categoria do Produto	Quantidade	Idade do Cliente	É Fraudulento	Idade da Conta em Dias	Hora da Transação
0	58.09	bank transfer	electronics	1.0	17.0	0	30	5
1	389.96	debit card	electronics	2.0	40.0	0	72	8
2	134.19	PayPal	home & garden	2.0	22.0	0	63	3
3	226.17	bank transfer	clothing	5.0	31.0	0	124	20
4	121.53	bank transfer	clothing	2.0	51.0	0	158	5

Figura 2.12: *Dataset* contendo os dados de transações fraudulentas em *e-commerces*

O *dataset* na figura acima mostra diversas características referentes às compras on-line como a quantidade, o método de pagamento, a categoria do produto, se é uma compra fraudulenta ou não dentre outras. Para o exemplo de remoção de outliers apenas a variável “Quantidade de Transações” será utilizada. A Figura 2.13 abaixo ilustra um gráfico de caixa para essa variável.

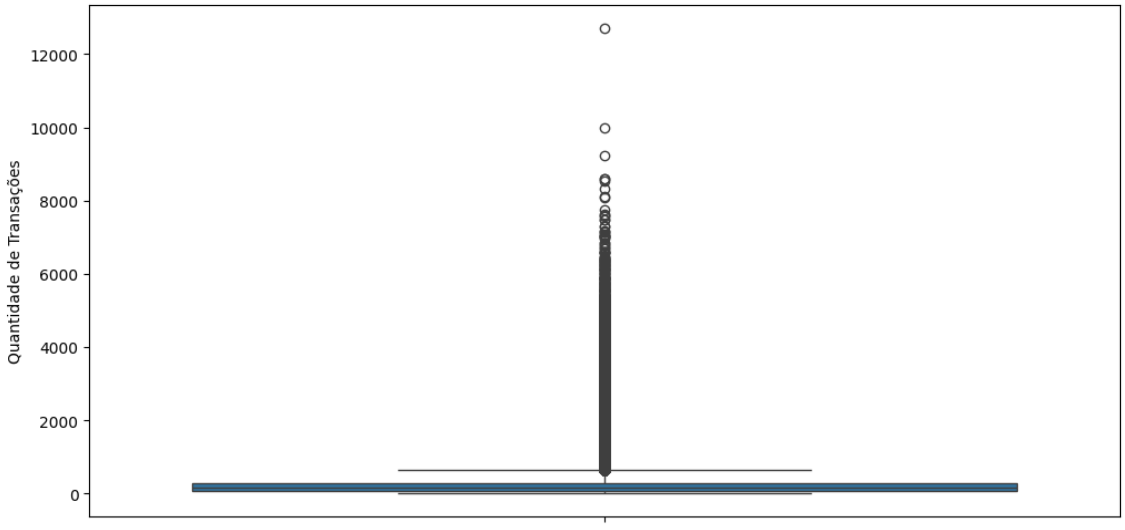

Figura 2.13: *Boxplot* para a variável “Quantidade de Transações” antes do tratamento de *outliers*

Seguindo o mesmo raciocínio exposto anteriormente é perceptível uma grande quantidade de outliers nessa variável. Existem muitas transações em quantidades acima dos milhares. O retângulo do *boxplot* é pouco perceptível. Para o tratamento dos outliers será utilizada a técnica de remoção interquartil. Nessa abordagem o primeiro quartil (Q1) e o terceiro quartil (Q3) são encontrados e, na sequência, a diferença entre eles é calculada (*interquartile range* - IQR). Após isso, um cálculo é realizado para estimar os valores que estão acima do limite superior ou abaixo do limite inferior. Tal cálculo utiliza um fator de multiplicação (normalmente 1,5 na maioria dos casos, sendo utilizado por diversos autores).

$$dados < Q1 - 1{,}5 * IQR \qquad \text{Equação 2.1}$$

$$dados > Q3 + 1{,}5 * IQR \qquad \text{Equação 2.2}$$

As equações acima definem os limites de corte para os dados, ou seja, dados que estão abaixo do resultado da expressão à direita na Equação 2.1 ou dados que estejam acima do resultado da expressão à direita na Equação 2.2 serão eliminados pois se tratam de *outliers*.

A Figura 2.14 abaixo apresenta o gráfico de caixa para a variável “Quantidade de Transações” após a remoção dos *outliers*.

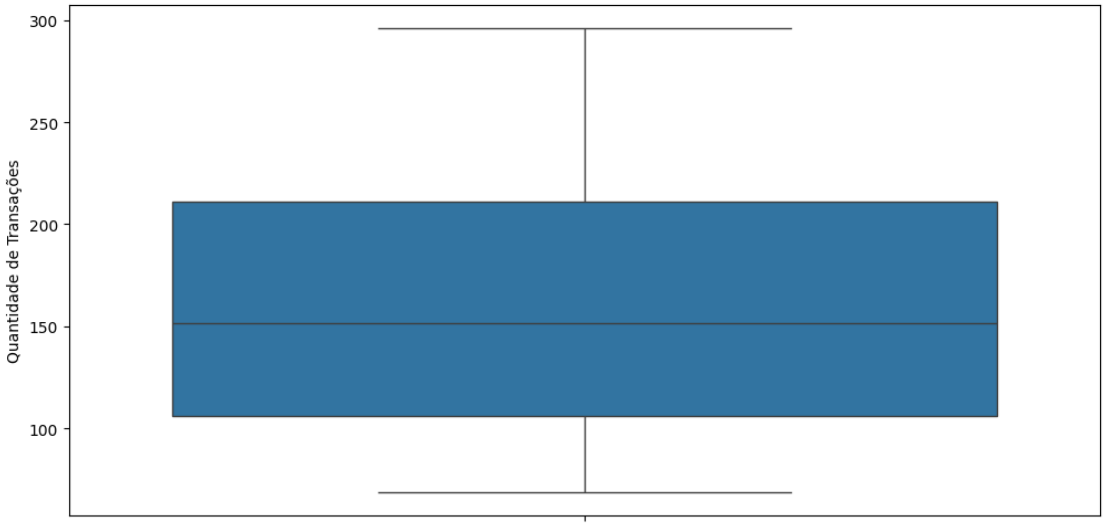

Figura 2.14: *Boxplot* para a variável “Quantidade de Transações” após o tratamento de *outliers*

Comparando as figuras 2.13 e 2.14 é possível notar uma grande diferença entre as distribuições dos dados. Dessa forma fica clara a efetividade do método para esse conjunto de dados. Vale destacar que após o a remoção dos outliers o *dataset* ficou com 751.255 registros. A Figura 2.15 abaixo apresenta um trecho de código contendo a metodologia para remoção de outliers interquartil. Foi implementada uma função por meio da palavra reservada *def* e na sequência uma cópia dos dados foi feita para finalmente a função ser chamada para remover os outliers da variável “Quantidade de Transações”.

```
def remover_outliers_interquartil(dados, variavel):

    Q1 = dados[variavel].quantile(q=0.25)
    Q3 = dados[variavel].quantile(q=0.75)
    IQR = dados[variavel].apply(stats.iqr)

    dados_limpos = dados[~((dados[variavel] < (Q1-1.5*IQR)) | (dados[variavel] > (Q3+1.5*IQR)))].copy()
    dados_limpos.reset_index(drop=True, inplace=True)

    return dados_limpos
```

```
df_fraud_clean = df_fraud.copy()
df_fraud_clean = remover_outliers_interquartil(df_fraud_clean, 'Quantidade de Transações')
```

Figura 2.15: Implementação do método de remoção de *outliers* interquartil

2.4. ANÁLISE GRÁFICA

No geral existem diversos tipos de representações gráficas para os dados e a escolha depende do ponto de vista que quer ser enfatizado. Novamente tomando como exemplo o *dataset* de transações fraudulentas apresentado inicialmente na Figura 2.12, pode-se utilizar histogramas para verificar a distribuição das variáveis numéricas. Veja a Figura 2.16 abaixo.

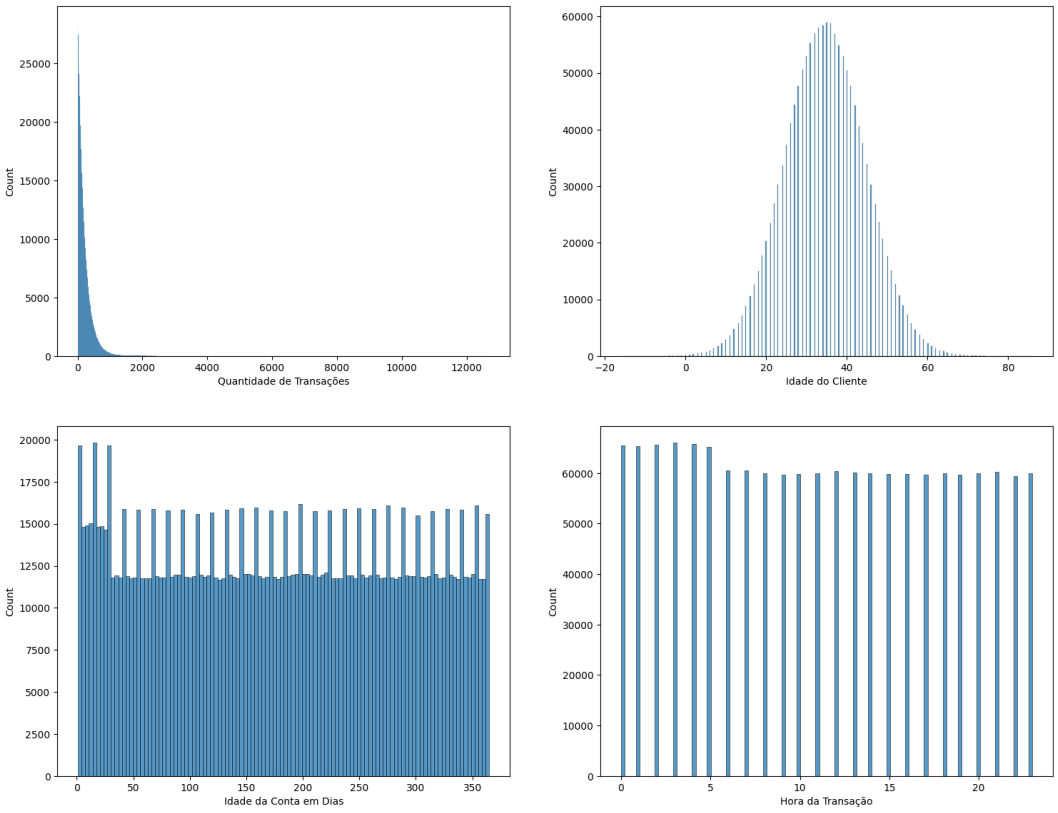

Figura 2.16: Histograma das variáveis numéricas contidas no *dataset* de transações fraudulentas

A variável “Quantidade de Transações” está concentrada em valores abaixo de 2000 como é possível ver na figura acima no quadro superior esquerdo. Além disso, uma informação interessante é a distribuição das idades que possui a maioria dos valores em torno de 35 a 40 anos como mostrado no quadro superior direito na figura acima. As variáveis “Idade da Conta em Dias” e “Hora da Transação” parecem ter um equilíbrio na distribuição dos dados, não

tendo nada que se destaque muito. A Figura 2.16 acima foi gerada por meio do trecho de código abaixo (Figura 2.17).

```
fig, ax = plt.subplots(2, 2, figsize=(18,14))
sns.histplot(data=df_fraud, x='Quantidade de Transações', ax=ax[0][0])
sns.histplot(data=df_fraud, x='Idade do Cliente', ax=ax[0][1])
sns.histplot(data=df_fraud, x='Idade da Conta em Dias', ax=ax[1][0])
sns.histplot(data=df_fraud, x='Hora da Transação', ax=ax[1][1])
```

Figura 2.17: Trecho de código mostrando a implementação dos histogramas.

As variáveis categóricas como “Método de Pagamento” e “Categoria do Produto” podem ser avaliadas por meio de gráficos de barras (*countplot*).

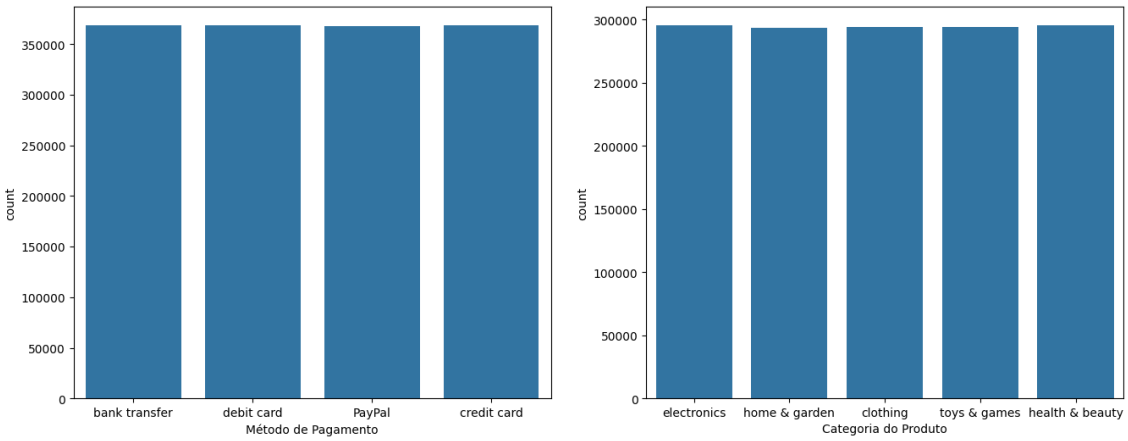

Figura 2.18: Gráficos de barras para as variáveis “Método de Pagamento” e “Categoria do Produto”

As duas variáveis possuem um equilíbrio em relação aos seus respectivos valores como mostrado na Figura 2.18, ou seja, no caso dos métodos de pagamento não existe uma prevalência de nenhum deles e no caso da variável categoria do produto todas vendem aproximadamente de forma equilibrada.

Uma outra visualização de dados muito interessante é o *pairplot* que apresenta as relações entre todas as variáveis numéricas no *dataset*. Veja a Figura 2.19 a seguir.

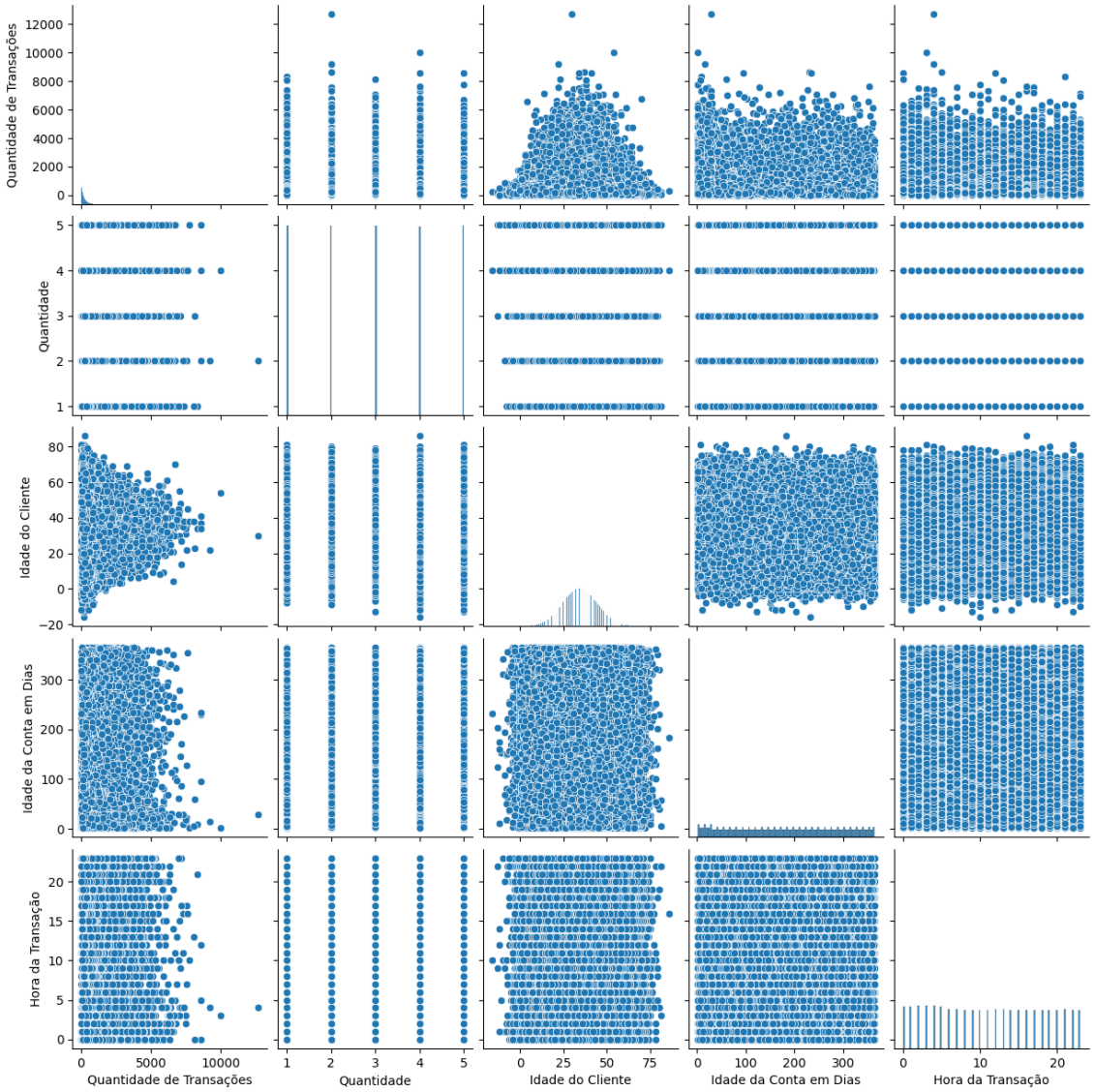

Figura 2.19: *Pairplot* das variáveis

Ao analisar a Figura 2.19 acima vemos que as variáveis “Idade do Cliente” e “Quantidade de Transações” apresentam uma relação interessante entre si. É possível ver na figura central da primeira linha que o aumento na idade implica em um aumento na quantidade de transações até um certo limite, e após isso, o aumento da idade passa a implicar em diminuição das transações. Nenhuma outra figura apresenta relação significativa.

UNIDADE 3 REDUÇÃO DE DIMENSIONALIDADE

Nesse momento já somos capazes de aplicar operações e transformações aos dados. Além disso já temos ferramentas para explorar e entender melhor as informações. Agora iremos trabalhar na redução de dimensionalidade.

OBJETIVOS DA UNIDADE 3

Ao final dos estudos, você deverá ser capaz de:

- Implementar um algoritmo de redução de dimensionalidade
- Compreender a importância de reduzir a dimensão dos dados

3.1. REDUÇÃO DE DIMENSIONALIDADE

A redução de dimensionalidade é um conceito utilizado em diversos conjuntos de dados no intuito de torná-los menos complexos. Essa é uma maneira de simplificar a representação dos dados e manter o máximo de informação possível. Além disso, a redução de dimensionalidade ajuda a remover informações redundantes. Em muitos casos, modelos de *machine learning* podem se tornar mais simples tanto do ponto de vista de quantidade de variáveis quanto do ponto de vista de custo computacional, visto que a remoção de características reduz o tempo de treinamento.

Existem diversas técnicas de redução de dimensionalidade com o Análise de Componentes Principais (PCA), Análise de Discriminante Linear (LDA), *t-Distributed Stochastic Neighbor Embedding* (t-SNE), *Autoencoders* dentre outras. Nesse material estudaremos a PCA, ficando a cargo do leitor buscar conhecer as demais técnicas.

3.2. ANÁLISE DE COMPONENTES PRINCIPAIS

A Análise de Componentes Principais (PCA) é uma técnica de redução de dimensionalidade muito utilizada. Dado um conjunto de dados (tabela/*dataset*), seu princípio de funcionamento está na combinação linear das colunas existentes. Dessa forma é possível escolher a quantidade de colunas resultantes dessa operação que passarão a ser as novas variáveis no *dataset*. As colunas resultantes possuem a maior variância possível com base na combinação linear realizada, dessa forma, a redução de variáveis preserva a maior parte da variabilidade dos dados originais. A PCA é extremamente útil nos casos em que existe uma quantidade muito grande de variáveis para determinado problema.

No intuito de ilustrar essa técnica será utilizado um conjunto de dados proveniente do Kaggle. Os dados em questão podem ser encontrados no *link* e se referem a indicadores socioeconômicos e fatores de saúde em diversos países. Os nomes das colunas foram traduzidos do inglês para o português. A Figura 3.1 a seguir apresenta os cinco primeiros registros do *dataset*.

	país	mortalidade_infantil	exportações	gastos_com_saúde	imports	renda	inflação	expectativa_de_vida	fertilidade_total	PIB_PER_CAPITA
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200

Figura 3.1: *Dataset* com indicadores socioeconômicos e fatores de saúde

Para implementar a PCA é interessante que os dados sejam transformados utilizando alguma das estratégias (MinMaxScaler, StandardScaler, ...) já mencionadas anteriormente. O trecho de código abaixo implementa de forma básica a aplicação da PCA a esse conjunto de dados reduzindo de 9 variáveis para 3.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

#Ler dados no diretório local
df = pd.read_csv('./Datasets/Clustering PCA/Country-data.csv')

#Renomear colunas
df.rename(columns={'country': 'país', 'child_mort':
'mortalidade_infantil', 'exports': 'exportações',
                   'health': 'gastos_com_saúde', 'imorts':
'importações', 'income': 'renda', 'inflation': 'inflação',
                   'life_expec': 'expectativa_de_vida', 'total_fer':
'fertilidade_total' , 'gdpp': 'PIB_PER_CAPITA'},
          inplace=True)

#Transformar os dados
scaler = MinMaxScaler()
data_transformed = scaler.fit_transform(df.drop('país', axis=1))
df_transformed = pd.DataFrame(data=data_transformed,
columns=df.drop('país', axis=1).columns)
df_transformed = pd.concat([df['país'], df_transformed], axis=1)

#PCA
pca = PCA(n_components=3)
X = pca.fit_transform(df_transformed.drop('país', axis=1))
df_pca = pd.DataFrame(data=X, columns=['component 1', 'component 2',
'component 3'])

print(df_pca)

#Plot 3D
fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(projection='3d')
ax.scatter(df_pca['component 1'], df_pca['component 2'],
df_pca['component 3'])
ax.set_xlabel('component 1')
ax.set_ylabel('component 2')
ax.set_zlabel('component 3')
fig.show()
```

A Figura 3.2 a seguir apresenta o *dataframe* resultante com as 3 componentes calculadas pela PCA. Dessa forma, saímos de um conjunto de dados que possuía 9 variáveis numéricas para um novo com apenas 3. Além disso, é possível ver um gráfico de dispersão de dados baseado nas três novas variáveis (Figura 3.3).

	component 1	component 2	component 3
0	-0.599078	0.095490	0.157554
1	0.158474	-0.212092	-0.064189
2	0.003686	-0.135867	-0.134182
3	-0.650235	0.275975	-0.142672
4	0.200711	-0.064662	-0.100715

Figura 3.2: *Dataframe* resultante após a aplicação da PCA

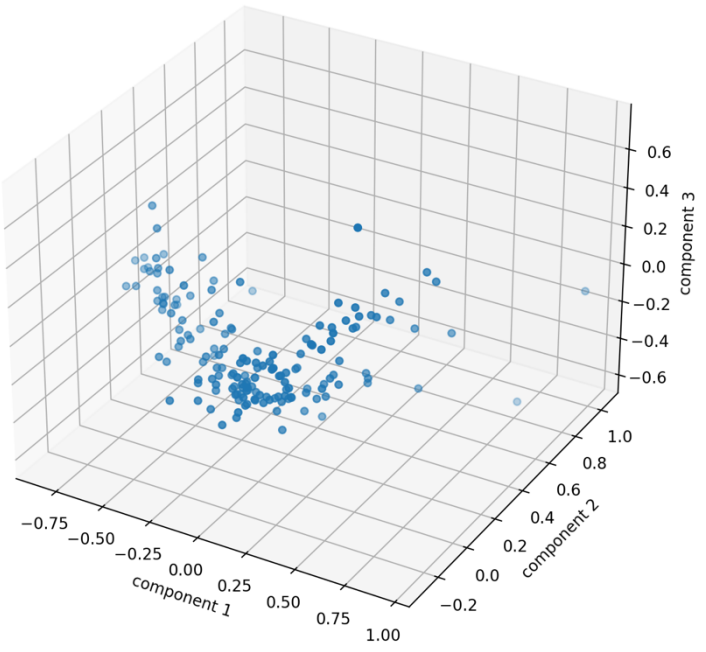

Figura 3.3: Representação em três dimensões das componentes calculadas pela PCA

Vale ressaltar que a PCA implementada pela biblioteca *Scikit-learn* é bastante flexível, de maneira que a quantidade de variáveis resultantes é indicada pelo parâmetro *n_components* no construtor do objeto PCA (trecho de código acima).

UNIDADE 4 CLUSTERIZAÇÃO

Nesta unidade, exploraremos os princípios da clusterização, que é uma tarefa comumente executada por modelos não-supervisionados. A clusterização, uma técnica fundamental no campo da mineração de dados, é dedicada à organização de conjuntos de dados em subgrupos ou "*clusters*", de modo que os elementos dentro de cada *cluster* sejam mais semelhantes entre si do que com elementos de outros *clusters*.

OBJETIVOS DA UNIDADE 4

Ao final dos estudos, você deverá ser capaz de:

- Implementar algoritmos de clusterização
- Analisar resultados e gerar visualizações de *clusters*

4.1. CLUSTERIZAÇÃO

A clusterização é uma técnica muito útil em mineração de dados visto que ajuda a agrupar dados semelhantes. Compreender quais dados são semelhantes entre si e a qual subgrupo pertencem é extremamente útil em diversas situações. Por exemplo, na área de varejo é possível segmentar clientes com base em suas preferências ou mesmo agrupar produtos semelhantes em um estoque. Umas das técnicas mais famosas de clusterização é o k-means que utiliza a distância entre os dados para definir os centros dos clusters e então agrupar os dados de maneira pertinente. É possível também agrupar os dados de forma hierárquica ou com base na densidade dos dados.

4.2. K-MEANS

O algoritmo K-means funciona com base na distância dos dados até os centros. Inicialmente são definidos k centros que na prática são a quantidade de clusters que se deseja configurar e na sequência o algoritmo ajusta os centros comparando as distâncias em relação aos dados. O valor de k a ser inserido, ou seja, a quantidade de clusters que se deseja ajustar é algo que pode variar e necessita de investigação. Para ajudar na definição de k existem alguns métodos como o *silhouette score* e o *elbow* (cotovelo). De forma resumida, ambos os métodos analisam a distância entre os pontos e os centros dos clusters e retornam valores ponderados. Portanto, no intuito de utilizar ambas as abordagens ajusta-se o algoritmo k-means diversas vezes, sendo que em cada tentativa um valor de k distinto é escolhido. Dessa forma, para cada execução o *silhouette* e o *elbow* são calculados e armazenados. No fim, cabe ao especialista analisar os resultados e decidir o valor de k.

Para ilustrar a implementação do k-means utilizaremos o mesmo conjunto de dados apresentado na seção anterior, especificamente na Figura 3.1 que apresenta indicadores socioeconômicos e fatores de saúde em diversos países. Como explicado anteriormente é necessário executar as análises para validar a quantidade de clusters (valor de k).

Foram testados os valores de 2 a 10 para k, e os resultados do *elbow* e *silhouette* foram armazenados. Especificamente para o método *elbow* é feito um gráfico que lembra um cotovelo (por isso o nome) no qual são inseridos os pares que contém o valor de k e a respectiva soma dos quadrados das distâncias (medida que efetivamente indica a qualidade da separação dos dados avaliando a distância até os centros dos clusters). A Figura 4.1 a seguir ilustra esse gráfico.

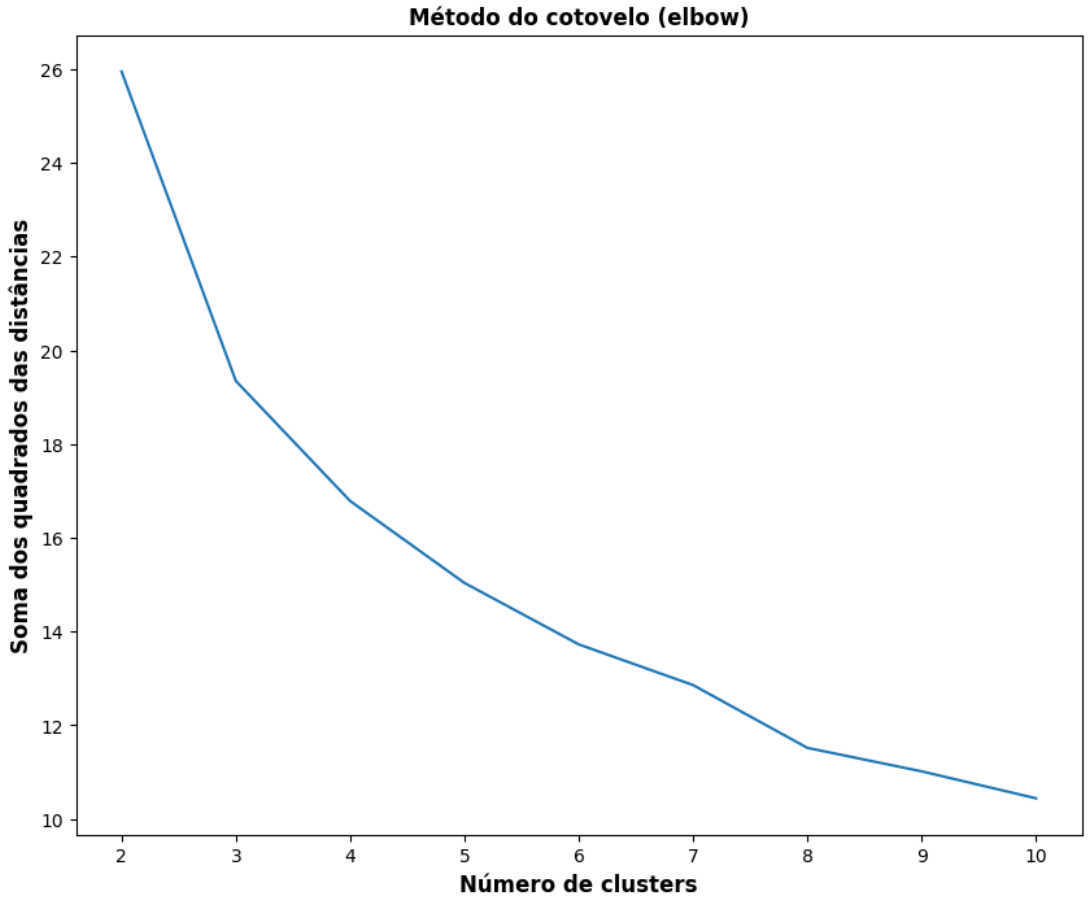

Figura 4.1: Curva representando o método do cotovelo

No geral, o que é buscado no gráfico do método do cotovelo é um ponto no qual a linha dobra de maneira mais acentuada (dando aparência de um cotovelo ao gráfico). Nem sempre fica claro qual é o ponto exato ao analisar o gráfico e por isso é interessante realizar alguns procedimentos matemáticos (Temporal, 2019).

Iremos traçar uma linha reta que liga os pontos das extremidades da curva do cotovelo e na sequência analisar na curva qual é o ponto mais distante da linha reta. Veja a Figura 4.2 a seguir que ilustra a situação.

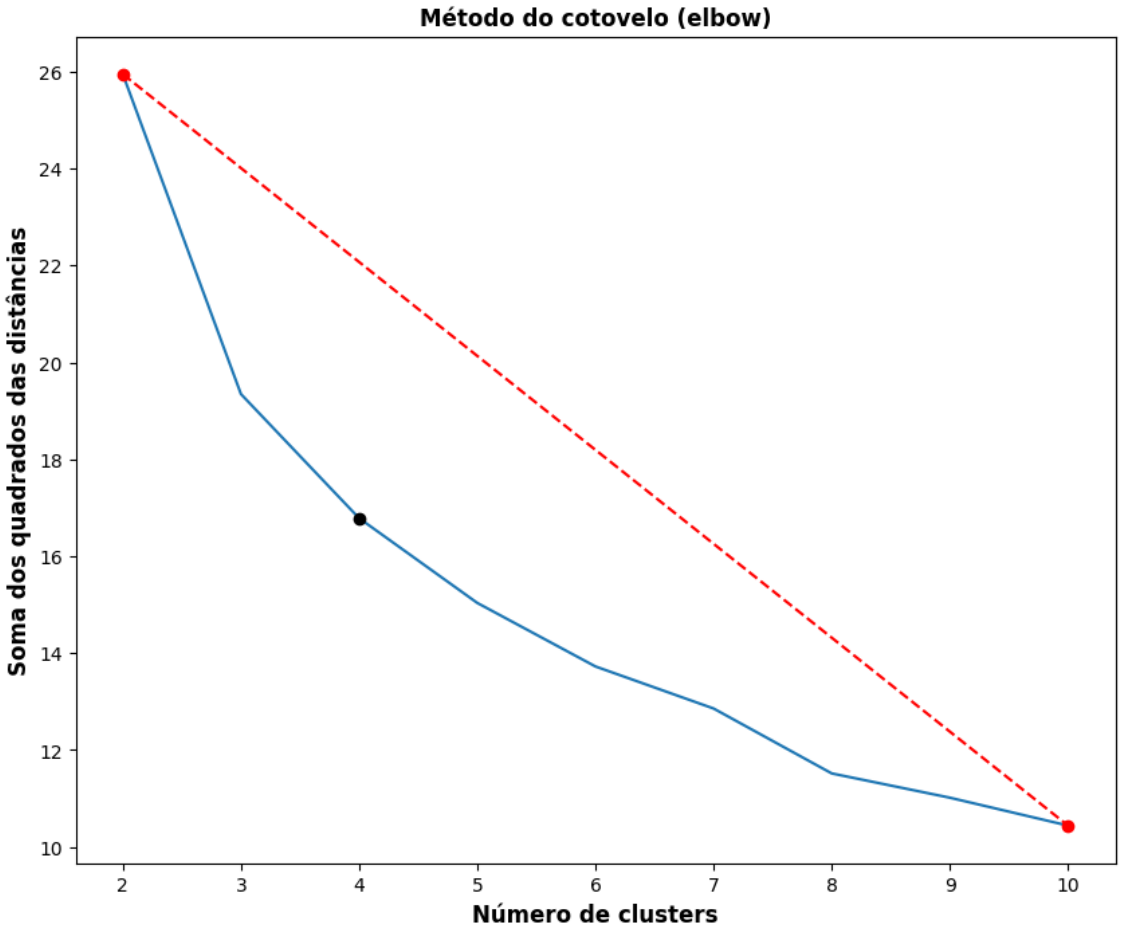

Figura 4.2: Curva representando o método do cotovelo e o cálculo do ponto mais distante da linha vermelha

Após executar o cálculo da distância de cada ponto na curva azul até a linha vermelha encontrou-se o ponto destacado em preto. Esse ponto tem coordenadas (4, 16.78) no qual 4 é o valor de k (quantidade de clusters) e 16.78 é o valor da soma dos quadrados das distâncias calculado pelo algoritmo. Portanto esse ponto é o mais distante da linha vermelha e, de acordo com o método do cotovelo é o melhor valor de k. Vale ressaltar que esse resultado é um indicativo interessante, mas não é a verdade absoluta. É sempre recomendado conduzir diversos testes e explorações nos dados até que se possa chegar a uma conclusão satisfatória.

Para fins didáticos, iremos seguir o resultado do método do cotovelo e utilizar k = 4. Dessa forma, teremos quatro clusters distintos. Para facilitar a visualização, aplicou-se uma PCA ao conjunto de dados, reduzindo de nove variáveis para duas no intuito de fazer um gráfico em duas dimensões (Figura 4.3).

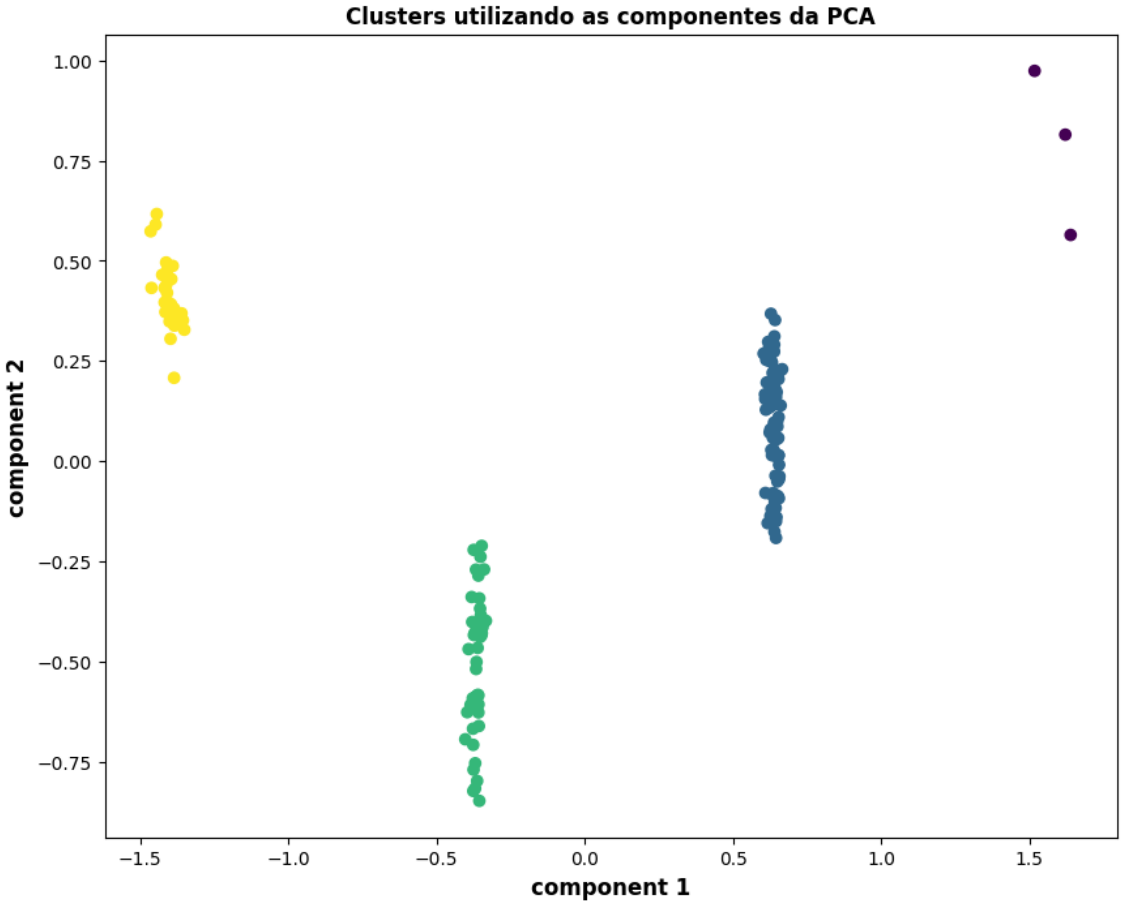

Figura 4.3: Representação gráfica dos quatro clusters utilizando as duas componentes da PCA

Aparentemente a divisão dos dados em quatro conjuntos parece satisfatória visto que os pontos estão bem organizados em partes distintas da Figura 4.3 acima. O trecho de código a seguir apresenta a implementação completa do k-means passando pela transformação dos dados, aplicação dos métodos *silhouette* e *elbow* e o *plot* dos gráficos.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

def numero_otimo_de_clusters(sse: dict) -> tuple[int, float]:

    primeira_chave = list(sse.keys())[0]
    ultima_chave = list(sse.keys())[-1]

    x1, y1 = primeira_chave, sse[primeira_chave]
    x2, y2 = ultima_chave, sse[ultima_chave]

    distancias = {}
    for i in sse.keys():
        if i != primeira_chave and i != ultima_chave:
            x0 = i
            y0 = sse[i]
            numerador = abs((y2 - y1) * x0 - (x2 - x1) * y0 + x2 * y1
- y2 * x1)
            denominador = np.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)
            distancias[i] = (numerador / denominador)

    return max(distancias, key=distancias.get), sse[max(distancias,
key=distancias.get)]

df = pd.read_csv('./Datasets/Clustering PCA/Country-data.csv')

df.rename(columns={'country': 'país', 'child_mort':
'mortalidade_infantil', 'exports': 'exportações',
                   'health': 'gastos_com_saúde', 'imorts':
'importações', 'income': 'renda', 'inflation': 'inflação',
                   'life_expec': 'expectativa_de_vida', 'total_fer':
'fertilidade_total', 'gdpp': 'PIB_PER_CAPITA'},
          inplace=True)

print('DADOS ORIGINAIS\n')
print(df.head())

scaler = MinMaxScaler()

data_transformed = scaler.fit_transform(df.drop('país', axis=1))
df_transformed = pd.DataFrame(data=data_transformed,
columns=df.drop('país', axis=1).columns)
df_transformed = pd.concat([df['país'], df_transformed], axis=1)

print('DADOS TRANSFORMADOS UTILIZANDO MINMAX\n')
print(df_transformed.head())

print('SELECIONAR A QUANTIDADE IDEAL DE CLUSTERS\n')

silhouette_dict = {}
sse = {}
```

```
for n in range(2, 11):
    kmeans = KMeans(n_clusters=n, random_state=0)
    silhouette_dict[n] = silhouette_score(df_transformed.drop('país',
axis=1).copy(),

kmeans.fit_predict(df_transformed.drop('país', axis=1).copy()))
    sse[n] = kmeans.inertia_

best_k_silhouette = max(silhouette_dict, key=silhouette_dict.get)
print(f'VALOR DE K UTILIZANDO SILHOUETTE: {best_k_silhouette}')
best_k_elbow, best_value_elbow = numero_otimo_de_clusters(sse)
print(f'VALOR DE K UTILIZANDO ELBOW: {best_k_elbow}')

primeira_chave = list(sse.keys())[0]
ultima_chave = list(sse.keys())[-1]
x_values = [primeira_chave, ultima_chave]
y_values = [sse[primeira_chave], sse[ultima_chave]]

plt.figure(figsize=(10,8))
plt.plot(sse.keys(), sse.values())
plt.plot(x_values, y_values, 'ro', linestyle='--')
plt.plot(best_k_elbow, best_value_elbow, 'ok')
plt.xlabel('Número de clusters', fontsize=12, fontweight='bold')
plt.ylabel('Soma dos quadrados das distâncias', fontsize=12,
fontweight='bold')
plt.title('Método do cotovelo (elbow)', fontsize=12,
fontweight='bold')
plt.savefig('./Elbow Method K-Means.png', bbox_inches='tight')
plt.close()

print(f'MELHOR PAR DE VALORES({best_k_elbow}, {best_value_elbow})\n')

kmeans = KMeans(n_clusters=best_k_elbow, random_state=0)
kmeans.fit_predict(df_transformed.drop('país', axis=1))

df_transformed['cluster'] = kmeans.labels_

print('DADOS TRANSFORMADOS E COM A COLUNA CLUSTER\n')
print(df_transformed.head())

pca = PCA(n_components=2)
X = pca.fit_transform(df_transformed.drop('país', axis=1))
df_pca = pd.DataFrame(data=X, columns=['component 1', 'component 2'])
df_pca['cluster'] = kmeans.labels_

plt.figure(figsize=(10,8))
plt.scatter(df_pca['component 1'], df_pca['component 2'],
c=df_pca['cluster'])
plt.xlabel('component 1', fontsize=12, fontweight='bold')
plt.ylabel('component 2', fontsize=12, fontweight='bold')
plt.title('Clusters utilizando as componentes da PCA', fontsize=12,
fontweight='bold')
plt.savefig('./Cluster Using PCA.png', bbox_inches='tight')
plt.close()
```

Em relação ao método *silhouette score*, valores mais próximos de 1 indicam uma melhor separação dos dados em relação àquela quantidade de clusters (k). Valores próximos de -1 são os piores possíveis e valores próximos de 0 indicam clusters que se sobrepõem. A Tabela 4.1 abaixo apresenta os valores de *silhouette* obtidos. O maior valor é 0.376714 referente à k=2. Portando, olhando para esse coeficiente, devemos trabalhar com dois clusters. Nesse momento fica uma dúvida referente aos dois métodos (silhouette e elbow) pois estão indicando valores distintos de k. Para obter um bom resultado é necessário conduzir uma exploração mais completa no conjunto de dados. Vale ressaltar que para fins didáticos implementamos o k-means utilizando k=4 já discutido anteriormente, mas poderíamos ter utilizado o valor k=2, ou mesmo conduzir mais análises a fim de averiguar qual o melhor resultado possível.

k	***silhouette***
2	0.376714
3	0.342655
4	0.346002
5	0.240452
6	0.233708
7	0.223302
8	0.243356
9	0.233399
10	0.217807

Tabela 4.1: Valores calculados do método silhouette

4.3. CLUSTER HIERÁRQUICO

Outra abordagem de clusterização bastante interessante é a aglomerativa (hierárquica) que agrupa os dados com base em sua similaridade. O processo é feito por meio de uma repetição analisando os dados de baixo para cima (bottom-up) de maneira a ir analisando as distâncias até que todos os dados tenham sido devidamente categorizados. Para ilustrar a implementação faremos uso da biblioteca *Scipy* (Virtanen et al., 2020) e dos mesmos dados da seção anterior. Aqui, para fins ilustrativos e didáticos, será feito um corte no dataset para utilizar apenas os vinte primeiros registros. Essa redução se deve pelo fato de que será apresentado um tipo de gráfico que ficaria muito poluído visualmente e pouco explicativo no caso em que fossem utilizados todos os dados disponíveis.

Para dar início ao processo, os dados são transformados (MinMaxScaler) e então removidos os vinte primeiros registros como já explicado anteriormente. A Figura 4.4 abaixo ilustra o *dataset* preparado para o processo de clusterização.

	mortalidade_infantil	exportações	gastos_com_saúde	imports	renda	inflação	expectativa_de_vida	fertilidade_total	PIB_PER_CAPITA
0	0.426485	0.049482	0.358608	0.257765	0.008047	0.126144	0.475345	0.736593	0.003073
1	0.068160	0.139531	0.294593	0.279037	0.074933	0.080399	0.871795	0.078864	0.036833
2	0.120253	0.191559	0.146675	0.180149	0.098809	0.187691	0.875740	0.274448	0.040365
3	0.566699	0.311125	0.064636	0.246266	0.042535	0.245911	0.552268	0.790221	0.031488
4	0.037488	0.227079	0.262275	0.338255	0.148652	0.052213	0.881657	0.154574	0.114242
5	0.057936	0.094006	0.390926	0.091610	0.145437	0.232049	0.861933	0.192429	0.096107
6	0.075463	0.103511	0.160970	0.260065	0.048967	0.110711	0.812623	0.085174	0.028529
7	0.010711	0.098509	0.430081	0.119782	0.327926	0.049626	0.984221	0.123028	0.493171
8	0.008277	0.256095	0.571162	0.274438	0.342396	0.046973	0.954635	0.045741	0.445447
9	0.178189	0.271103	0.252952	0.118632	0.123731	0.166436	0.729783	0.121451	0.053537
10	0.054528	0.174550	0.377874	0.250866	0.179201	0.035274	0.822485	0.111987	0.265050
11	0.029211	0.347144	0.196395	0.292261	0.325514	0.107661	0.865878	0.159306	0.195373
12	0.227848	0.079498	0.106277	0.124956	0.014720	0.104889	0.755424	0.186120	0.005030
13	0.056475	0.197062	0.382846	0.279612	0.118103	0.041872	0.879684	0.099369	0.150512
14	0.014119	0.256595	0.236172	0.370451	0.125339	0.178449	0.755424	0.053628	0.055350
15	0.009250	0.381663	0.552517	0.429094	0.325514	0.056279	0.944773	0.111987	0.421585
16	0.078870	0.290613	0.210690	0.330206	0.058453	0.049441	0.775148	0.246057	0.039220
17	0.527751	0.118520	0.142324	0.213495	0.009735	0.047084	0.585799	0.664038	0.005030
18	0.195229	0.212071	0.210690	0.406097	0.046716	0.094261	0.788955	0.194006	0.018603
19	0.214216	0.205567	0.188316	0.196822	0.038596	0.120044	0.779093	0.323344	0.016694

Figura 4.4: *Dataset* transformado contendo os vinte primeiros registros do conjunto original

Utilizando a biblioteca *Scipy* obtemos o dendrograma que ilustra como os dados estão estruturados (Figura 4.5).

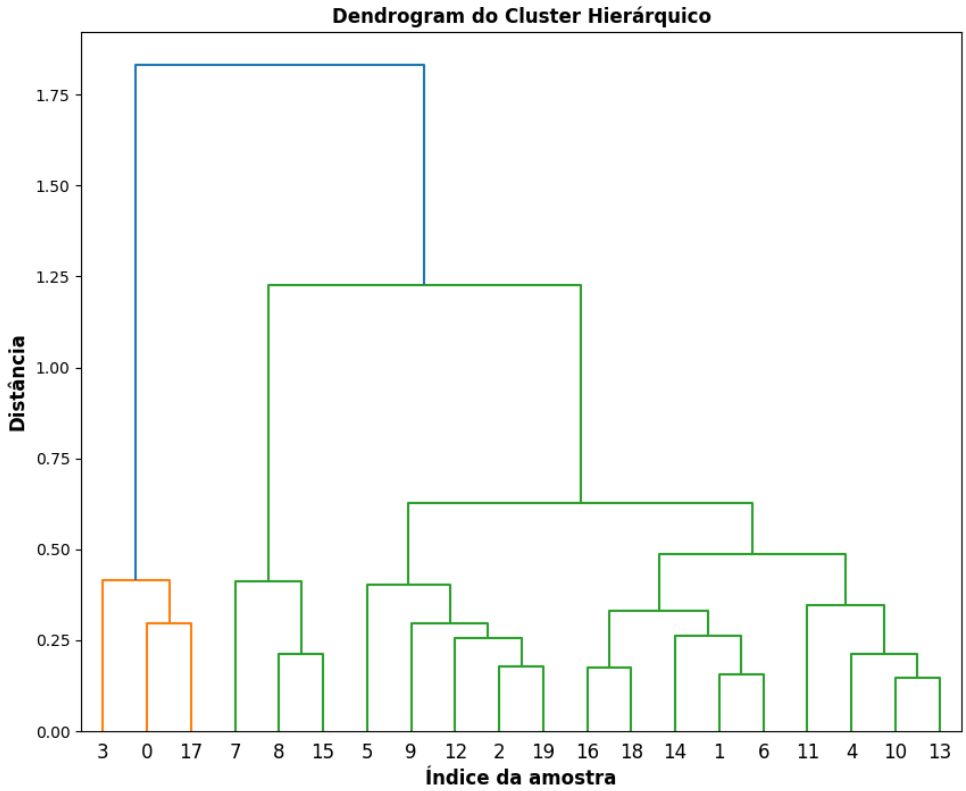

Figura 4.5: Dendrograma mostrando os níveis de agrupamento dos dados

Na dendrograma acima, cada folha (pontos na base da figura) representa uma amostra de dados (dentre as 20 que foram selecionadas) e no topo (raiz) a junção representa o *dataset* completo. A distância (eixo y) indica o quão similares os clusters são entre si. É interessante analisar nessa figura as ramificações que indicam exatamente como os dados se agrupam. As cores juntamente com as ramificações no dendrograma indicam de maneira clara os dados mais similares e a estruturação dos clusters. Para definir a quantidade de clusters por meio do dendrograma passa-se uma linha horizontal em determinada altura. Cada linha que cruzar essa horizontal será um cluster distinto.

Aqui vale ressaltar novamente a discussão com o time e a reflexão com base nos dados, visto que a altura dessa linha horizontal (distância) indica exatamente a similaridade entre os dados. Mantendo o foco didático do nosso texto, faremos uma linha horizontal na altura de 0,75. Observando a Figura 4.5, é possível perceber que traçar a linha horizontal nessa altura determina 3 clusters, o primeiro possui as amostras 3, 0 e 17, o segundo possui as amostras 7, 8 e 15 e o terceiro as restantes (5, 9, 12, 2, 19, 16, 18, 14, 1, 6, 11, 4, 10, 13). A Figura 4.6 abaixo mostra o *dataset* com a coluna cluster mais à direita.

	mortalidade_infantil	exportações	gastos_com_saúde	imports	renda	inflação	expectativa_de_vida	fertilidade_total	PIB_PER_CAPITA	cluster
0	0.426485	0.049482	0.358608	0.257765	0.008047	0.126144	0.475345	0.736593	0.003073	1
1	0.068160	0.139531	0.294593	0.279037	0.074933	0.080399	0.871795	0.078864	0.036833	3
2	0.120253	0.191559	0.146675	0.180149	0.098809	0.187691	0.875740	0.274448	0.040365	3
3	0.566699	0.311125	0.064636	0.246266	0.042535	0.245911	0.552268	0.790221	0.031488	1
4	0.037488	0.227079	0.262275	0.338255	0.148652	0.052213	0.881657	0.154574	0.114242	3
5	0.057936	0.094006	0.390926	0.091610	0.145437	0.232049	0.861933	0.192429	0.096107	3
6	0.075463	0.103511	0.160970	0.260065	0.048967	0.110711	0.812623	0.085174	0.028529	3
7	0.010711	0.098509	0.430081	0.119782	0.327926	0.049626	0.984221	0.123028	0.493171	2
8	0.008277	0.256095	0.571162	0.274438	0.342396	0.046973	0.954635	0.045741	0.445447	2
9	0.178189	0.271103	0.252952	0.118632	0.123731	0.166436	0.729783	0.121451	0.053537	3
10	0.054528	0.174550	0.377874	0.250866	0.179201	0.035274	0.822485	0.111987	0.265050	3
11	0.029211	0.347144	0.196395	0.292261	0.325514	0.107661	0.865878	0.159306	0.195373	3
12	0.227848	0.079498	0.106277	0.124956	0.014720	0.104889	0.755424	0.186120	0.005030	3
13	0.056475	0.197062	0.382846	0.279612	0.118103	0.041872	0.879684	0.099369	0.150512	3
14	0.014119	0.256595	0.236172	0.370451	0.125339	0.178449	0.755424	0.053628	0.055350	3
15	0.009250	0.381663	0.552517	0.429094	0.325514	0.056279	0.944773	0.111987	0.421585	2
16	0.078870	0.290613	0.210690	0.330206	0.058453	0.049441	0.775148	0.246057	0.039220	3
17	0.527751	0.118520	0.142324	0.213495	0.009735	0.047084	0.585799	0.664038	0.005030	1
18	0.195229	0.212071	0.210690	0.406097	0.046716	0.094261	0.788955	0.194006	0.018603	3
19	0.214216	0.205567	0.188316	0.196822	0.038596	0.120044	0.779093	0.323344	0.016694	3

Figura 4.6: *Dataset* com a coluna indicando o cluster de cada amostra

O trecho de código abaixo ilustra a implementação completa da abordagem de clusterização hierárquica.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

df = pd.read_csv('./Datasets/Clustering PCA/Country-data.csv')

df.rename(columns={'country':'país',
'child_mort':'mortalidade_infantil', 'exports':'exportações',
                   'health':'gastos_com_saúde', 'imorts':'importações',
'income':'renda', 'inflation':'inflação',
                   'life_expec':'expectativa_de_vida',
'total_fer':'fertilidade_total' , 'gdpp':'PIB_PER_CAPITA'},
          inplace=True)

scaler = MinMaxScaler()

data_transformed = scaler.fit_transform(df.drop('país', axis=1))
df_transformed = pd.DataFrame(data=data_transformed,
columns=df.drop('país', axis=1).columns)
df_transformed = pd.concat([df['país'], df_transformed], axis=1)

X_cluster = df_transformed.drop('país', axis=1).iloc[:20].copy()
Z = linkage(X_cluster, 'ward')

max_d = 0.75
clusters = fcluster(Z, max_d, criterion='distance')
X_cluster['cluster'] = clusters

plt.figure(figsize=(10,8))
dendrogram(Z)
plt.title('Dendrogram do Cluster Hierárquico', fontsize=12,
fontweight='bold')
plt.xlabel('Índice da amostra', fontsize=12, fontweight='bold')
plt.ylabel('Distância', fontsize=12, fontweight='bold')
plt.savefig('./Dendograma_Cluster_Hierarquico.png')

pca = PCA(n_components=2)
X = pca.fit_transform(X_cluster.drop('cluster', axis=1))
df_pca = pd.DataFrame(data=X, columns=['component 1', 'component 2'])
df_pca['cluster'] = clusters

plt.figure(figsize=(10,8))
plt.scatter(df_pca['component 1'], df_pca['component 2'],
c=df_pca['cluster'])
plt.xlabel('component 1', fontsize=12, fontweight='bold')
plt.ylabel('component 2', fontsize=12, fontweight='bold')
plt.title('Clusters utilizando as componentes da PCA', fontsize=12,
fontweight='bold')
plt.savefig('./Cluster Hierarquico Using PCA.png',
bbox_inches='tight')
```

UNIDADE 5 REGRAS DE ASSOCIAÇÕES

Nessa unidade faremos o estudo de modelos que são capazes de analisar relações entre os dados. Essas relações são também conhecidas como padrões frequentes e estamos interessados em minerar/descobrir tais relações.

OBJETIVOS DA UNIDADE 5

Ao final dos estudos, você deverá ser capaz de:

- Implementar novos modelos capazes de analisar padrões frequentes
- Compreender a importância dos padrões frequentes em conjuntos de dados

5.1. MINERAÇÃO DE PADRÕES FREQUENTES

A mineração de padrões frequentes abrange técnicas que analisam conjuntos de dados em busca de padrões que se repetem. É amplamente utilizado em loja virtuais (*e-commerce*) com o intuito de compreender os comportamentos de compra e recomendar produtos aos clientes. Dessa forma, é possível compreender quais produtos são frequentemente adquiridos em conjunto. Tais técnicas podem ser utilizadas também para analisar padrões fraudulentos que se assemelham e ocorrem em conjunto. Apriori e Fpgrowth (Faceli et al., 2023)são dois algoritmos comuns e bastante utilizados para a tarefa de mineração de padrões frequentes. Iremos ilustrar em mais detalhes suas implementações nas próximas seções por meio da biblioteca MLxtend (Raschka, 2018).

5.1. APRIORI

O algoritmo Apriori foi o primeiro a surgir com o objetivo de minerar padrões frequentes e utiliza uma estratégia de busca em largura (Faceli et al., 2023). Para exemplificar sua utilização iremos utilizar um conjunto de dados de vendas de jogos de videogame obtido no Kaggle. Esse conjunto possui 5918 registros de vendas de jogos e seus primeiros cinco registros podem ser vistos na Figura 5.1 abaixo.

	Coluna 1	Coluna 2	Coluna 3	Coluna 4	Coluna 5	Coluna 6
0	Left 4 Dead	Assassin's Creed 2	Super Mario World	The Last of Us	Read Dead Redemption	The Elder Scrolls V: Skyrim
1	Left 4 Dead	Minecraft	The Last of Us	Dark Souls	Read Dead Redemption	Resident Evil 4
2	Dark Souls	Assassin's Creed 2	Grand Theft Auto V	Resident Evil 4	Super Mario World	Guitar Hero 3
3	Resident Evil 4	Assassin's Creed 2	Read Dead Redemption	Minecraft	Dark Souls	Left 4 Dead
4	Grand Theft Auto V	The Last of Us	Guitar Hero 3	Minecraft	God of War	Dark Souls

Figura 5.1: *Dataset* contendo vendas de jogos de videogame

Na figura acima, cada linha representa uma venda distinta e o nosso objetivo com o algoritmo Apriori é analisar os padrões dos dados e verificar quais jogos são frequentemente comprados em conjunto. Interessante destacar que a biblioteca MLxtend possui um *encoder* que transforma o *dataframe* de maneira a indicar os nomes dos elementos (nesse exemplo jogos) nas colunas e sua presença ou ausência naquela transação por meio de variáveis *booleanas* (*True* ou *False*). Veja a Figura 5.2 abaixo que é o resultado da transformação da Figura 5.1.

	Assassin's Creed 2	Dark Souls	God of War	Grand Theft Auto V	Guitar Hero 3	Left 4 Dead	Minecraft	Read Dead Redemption	Resident Evil 4	Super Mario World	The Elder Scrolls V: Skyrim	The Last of Us
0	True	False	False	False	False	True	False	True	False	True	True	True
1	False	True	False	False	False	True	True	True	True	False	False	True
2	True	True	False	True	True	False	False	False	True	True	False	False
3	True	True	False	False	False	True	True	True	True	False	False	False
4	False	True	True	True	True	False	True	False	False	False	False	True

Figura 5.2: *Dataset* transformado utilizando variáveis *booleanas*

Finalmente o conjunto de dados está preparado para ser aplicado ao algoritmo Apriori. Basta chamar o método passando o *dataset* e o parâmetro de suporte mínimo (varia de 0 a 1). Esse parâmetro é muito importante visto que ele define o percentual de vezes que determinados subconjuntos aparecem em relação ao total de dados. Por exemplo, se o valor do suporte mínimo for de 0,05, estamos informando ao algoritmo que estamos interessados em encontrar padrões que se repitam pelo menos 5% em relação ao total de dados. Em nosso dataset de jogos possuímos 5918 amostras. Olhando para 5% dos dados, implica em aproximadamente 296 registros nos quais o mesmo padrão de compra de jogos aparece. Vale ressaltar novamente que o entendimento dos dados é muito importante, e algoritmos de mineração de padrões como Apriori são ferramentas poderosas. Em muitos casos, valores de suporte mínimo muito altos (0,7 por exemplo) podem não retornar resultados, visto que provavelmente não existe um percentual tão alto de qualquer padrão naquele conjunto de dados.

Após a execução do algoritmo temos um novo *dataframe* contendo os padrões encontrados (*itemsets*) e os respectivos valores de suporte (Figura 5.3).

	support	itemsets
0	0.482595	(Assassin's Creed 2)
1	0.506083	(Dark Souls)
2	0.500338	(God of War)
3	0.501014	(Grand Theft Auto V)
4	0.504224	(Guitar Hero 3)
...	...	...
293	0.089388	(The Elder Scrolls V: Skyrim, Read Dead Redemp...
294	0.093782	(Super Mario World, Resident Evil 4, The Elder...
295	0.084657	(Super Mario World, The Last of Us, Resident E...
296	0.089895	(The Elder Scrolls V: Skyrim, The Last of Us, ...
297	0.094120	(Super Mario World, The Last of Us, The Elder ...

Figura 5.3: Resultado do algoritmo Apriori

A resposta possui muitas linhas e se torna inviável exibir todas elas, além disso, os nomes dos jogos são grandes, o que dificulta ver os *itemsets* completos nos índices finais (a partir de 293). O resultado, no entanto, é bem completo, visto que nesse *dataframe* é possível ver o suporte mínimo para cada *itemset* e quais são os jogos adquiridos frequentemente juntos. Para melhorar a visualização e com intuito ilustrativo será utilizado o comando *explode()* da biblioteca Pandas nas linhas 296 e 297 para separar os itens e melhorar a visualização (Figura 5.4).

	support	itemsets
296	0.089895	The Elder Scrolls V: Skyrim
296	0.089895	The Last of Us
296	0.089895	Resident Evil 4
297	0.094120	Super Mario World
297	0.094120	The Last of Us
297	0.094120	The Elder Scrolls V: Skyrim

Figura 5.4: Itens expandidos para os índices 296 e 297 do resultado do algoritmo Apriori

É possível perceber que The Elder Scrolls V: Skyrim, The Last of Us e Resident Evil 4 são frequentemente adquiridos juntos com um suporte mínimo de aproximadamente 8%. Super Mario World, The Last of Us e The Elder Scrolls V: Skyrim possuem um suporte mínimo de aproximadamente 9%. Como visto da Figura 5.3 os resultados são extensos e necessitam de análise cuidadosa, porém, o objetivo desse texto foi apenas ilustrar e clarificar o uso do algoritmo, não sendo possível analisar o resultado em sua completude. O trecho de código abaixo apresenta a implementação completa do algoritmo Apriori. Algumas modificações nos nomes das colunas foram conduzidas para deixar o conjunto de dados mais simples e legível.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

df = pd.read_csv('./Datasets/games_sales_dataset.csv')

df.drop(['Unnamed: 6', 'Unnamed: 7', 'Unnamed: 8', 'Unnamed: 9',
'Unnamed: 10', 'Unnamed: 11'],
        axis=1, inplace=True)
df.dropna(inplace=True, ignore_index=True)

df.rename(columns={'God of War':'Coluna 1', 'The Last of Us':'Coluna
2', 'Read Dead Redemption':'Coluna 3',
                   'Minecraft':'Coluna 4', 'Grand Theft Auto
V':'Coluna 5', 'Left 4 Dead':'Coluna 6'},
                   inplace=True)

te = TransactionEncoder()
te_array = te.fit_transform(df.values)
df_transformed = pd.DataFrame(data=te_array, columns=te.columns_)

print('DADOS TRANSFORMADOS PELO ENCODER\n')
print(df_transformed.head())

df_item_frequentes_apriori = apriori(df_transformed, min_support=0.05,
use_colnames=True)

print('ITENS FREQUENTES\n')
print(df_item_frequentes_apriori)
```

5.2. FP-GROWTH

O algoritmo FP-growth utiliza a estratégia de busca por profundidade em árvores para minerar os padrões frequentes. Basicamente o método constrói uma árvore para armazenar as regras de associação relativas aos dados (Faceli et al., 2023). Da mesma maneira que o Apriori é um algoritmo bastante utilizado e eficaz na busca por padrões frequentes. Para ilustrar seu funcionamento será utilizado o mesmo conjunto de dados de jogos de videogame previamente discutido. Além disso, a estratégia de *encoder* dos dados é a mesma já vista para o Apriori. Na prática o que muda é a chamada do algoritmo em si, pois os demais passos são iguais aos já apresentados. Veja o trecho de código abaixo.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth

df = pd.read_csv('./Datasets/games_sales_dataset.csv')

df.drop(['Unnamed: 6', 'Unnamed: 7', 'Unnamed: 8', 'Unnamed: 9',
'Unnamed: 10', 'Unnamed: 11'],
        axis=1, inplace=True)
df.dropna(inplace=True, ignore_index=True)

df.rename(columns={'God of War':'Coluna 1', 'The Last of Us':'Coluna
2', 'Read Dead Redemption':'Coluna 3',
                   'Minecraft':'Coluna 4', 'Grand Theft Auto
V':'Coluna 5', 'Left 4 Dead':'Coluna 6'},
                   inplace=True)

te = TransactionEncoder()
te_array = te.fit_transform(df.values)
df_transformed = pd.DataFrame(data=te_array, columns=te.columns_)

print('DADOS TRANSFORMADOS PELO ENCODER\n')
print(df_transformed.head())

df_item_frequentes_fpgrowth = fpgrowth(df_transformed,
min_support=0.05, use_colnames=True)

print('ITENS FREQUENTES\n')
print(df_item_frequentes_fpgrowth)
```

	support	itemsets
0	0.507435	(Read Dead Redemption)
1	0.501183	(The Last of Us)
2	0.500507	(The Elder Scrolls V: Skyrim)
3	0.500507	(Super Mario World)
4	0.491213	(Left 4 Dead)
...	...	...
293	0.088374	(The Elder Scrolls V: Skyrim, God of War, Resi...
294	0.090571	(The Elder Scrolls V: Skyrim, God of War, Supe...
295	0.090064	(The Elder Scrolls V: Skyrim, Dark Souls, God ...
296	0.089388	(The Elder Scrolls V: Skyrim, Read Dead Redemp...
297	0.093106	(The Elder Scrolls V: Skyrim, Guitar Hero 3, G...

Figura 5.5: Resultado do algoritmo FP-growth

A Figura 5.5 acima ilustra os resultados do algoritmo FP-growth que parecem ser um pouco diferentes em relação ao Apriori. Abaixo seguem os últimos itens expandidos para uma melhor compreensão.

	support	itemsets
296	0.089388	The Elder Scrolls V: Skyrim
296	0.089388	Read Dead Redemption
296	0.089388	God of War
297	0.093106	The Elder Scrolls V: Skyrim
297	0.093106	Guitar Hero 3
297	0.093106	God of War

Figura 5.6: Itens expandidos para os índices 296 e 297 do resultado do algoritmo FP-growth

É possível ver que utilizando o FP-growth houve mudanças nos padrões extraídos. Para esses dois últimos registros (296 e 297) apareceram os jogos Guitar Hero 3, God of War e Read Dead Redemption. Tais mudanças podem ser simplesmente na ordem dos índices e o Apriori pode ter encontrado os mesmos padrões, mas não exatamente nas últimas posições do *dataframe* de resultados. Reforço aqui sempre inspecionar com cuidado os resultados e compreender o contexto geral dos dados.

FINALIZAR

Após a leitura desse material, espero que o aprendizado dos conceitos relacionados à mineração de dados esteja mais fácil. Como dito anteriormente, essa foi uma introdução ao assunto e abrirá caminho para diversos outros conceitos.

As técnicas utilizadas nesse material são básicas e de caráter educativo, visto que o leitor precisa se familiarizar com os conteúdos e amadurecer aos poucos para conseguir compreender tópicos mais complexos. Os modelos de mineração de dados apresentados aqui foram todos utilizados com suas configurações padrão. O leitor pode buscar se aprofundar em técnicas que buscam os melhores parâmetros dentro de um conjunto definido para cada modelo.

Além disso, várias questões mais profundas relacionadas aos modelos não foram abordadas visto que são um pouco mais complexas para esse início de jornada.

Espero que esse material tenha despertado a curiosidade no leitor para continuar investigando esse mundo maravilhoso dos dados. Parabenizo o esforço empregado até aqui e convido o leitor a mergulhar de cabeça nos estudos pois vale a pena.

Prof. Dr. Thiago Santana Lemes

Sobre o autor

Thiago Santana Lemes é doutor em Engenharia Elétrica e de Computação pela Universidade Federal de Goiás (UFG), Cientista de dados na Ernst Young (EY). Experiência docente no ensino básico atuando na disciplina de matemática e nos cursos de Engenharia e Sistemas de Informação atuando nas disciplinas de cálculo diferencial, álgebra linear, cálculo numérico, equações diferenciais, probabilidade e estatística, algoritmos dentre outras. Desenvolve pesquisas na área de inteligência artificial com modelos de Machine Learning (implementação de modelos de regressão, classificação e recomendação), Deep Learning (redes convolucionais, redes recorrentes) e modelos de otimização (algoritmos genéticos e modelos lineares).

Referências Bibliográficas

Faceli, K., Lorena, A. C., Gama, J., Almeida, T. A. de, & Carvalho, A. C. P. L. F. de. (2023). *Inteligência Artificial Uma Abordagem de Aprendizado de Máquina* (2nd ed.). LTC.

Goldschmidt, R., & Passos, E. (2005). *Data Mining Um Guia Prático*. Elsevier.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Pedregosa, F. and V., G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, & M. and Duchesnay. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Raschka, S. (2018). MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack. *Journal of Open Source Software*, *3*(24), 638. https://doi.org/10.21105/joss.00638

Temporal, J. (2019). *Como definir o número de clusters para o seu KMeans*. Https://Medium.Com/Pizzadedados/Kmeans-e-Metodo-Do-Cotovelo-94ded9fdf3a9.

The pandas development team. (2024). *Pandas*. Https://Pandas.Pydata.Org.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Vázquez-Baeza, Y. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Waskom, M. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021