# Capítulo 3

# Ciência de Dados em Saúde: Primeiros Passos na Preparação e Análise de Dados

Ivan Rodrigues de Moura, Francisco José da Silva e Silva, Luciano Reis Coutinho, Ariel Soares Teles, Nailton dos Reis Maciel e Danilo Gameleira Dias

***Abstract***

*Data preparation and exploratory analysis are essential stages in data science applied to healthcare, ensuring the integrity, reliability, and relevance of the information used in clinical, epidemiological, and other biomedical applications. This chapter presents a set of fundamental techniques for organizing, cleaning, and exploring health-related data, focusing on biomedical and epidemiological datasets. Topics covered include data reading, structuring, cleaning, handling of missing and duplicate values, data manipulation and aggregation, and data visualization methods. The tools discussed include widely used libraries in the Python ecosystem, such as Pandas, Seaborn, and Plotly, offering a practical and accessible foundation for exploratory data analysis in healthcare. The content is intended for students and professionals in technology and healthcare, including those with no prior experience in data science, aiming to support their introduction to the initial data analysis processes in this interdisciplinary domain.*

***Resumo***

*A preparação de dados e a análise exploratória constituem etapas essenciais na ciência de dados aplicada à saúde, pois asseguram a integridade, a confiabilidade e a relevância das informações utilizadas em estudos clínicos, epidemiológicos e outras aplicações biomédicas. Este capítulo apresenta um conjunto de técnicas fundamentais para a organização, limpeza e exploração de dados de saúde, com foco em conjuntos de dados biomédicos e epidemiológicos. São abordados tópicos como leitura, estruturação e higienização de dados, tratamento de valores ausentes e duplicados, manipulação e agregação de informações, bem como métodos de visualização de dados. As ferramentas discutidas incluem bibliotecas amplamente utilizadas no ecossistema Python, como Pandas, Seaborn e Plotly, proporcionando uma base prática e acessível para a análise exploratória de*

*dados em saúde. O conteúdo é direcionado a estudantes e profissionais das áreas de tecnologia e saúde, mesmo aqueles sem experiência prévia em ciência de dados, e tem como objetivo facilitar a introdução aos processos iniciais de análise de dados neste domínio interdisciplinar.*

## 3.1. Introdução

Atualmente, a grande quantidade de dados providos do setor de saúde oferece uma janela de oportunidades significativa para aprimorar diagnósticos, personalizar tratamentos e otimizar a gestão hospitalar [Chattu 2021]. No entanto, características como a complexidade e heterogeneidade dos dados impõem desafios significativos à análise e interpretação eficaz dessas informações, exigindo soluções avançadas como análise de dados e ciência de dados [Awrahman et al. 2022]. A ciência de dados, combinando métodos estatísticos, aprendizado de máquina e visualização de dados, emerge como uma ferramenta essencial para extrair conhecimento útil a partir dessas grandes bases de dados, com potencial para transformar desde diagnósticos médicos até a gestão hospitalar [Subrahmanya et al. 2022]. Por exemplo, modelos de aprendizado de máquina podem ser utilizados para predizer o desenvolvimento de doenças como diabetes, depressão e hipertensão [Moura et al. 2023]. Além disso, técnicas de análise de dados e visualização de dados podem ser utilizadas para identificar padrões úteis na tomada de decisões médicas [Subrahmanya et al. 2022].

Diante desse cenário, este capítulo de livro tem como objetivo fornecer uma introdução às principais ferramentas e técnicas da ciência de dados aplicadas à saúde, com ênfase na preparação e análise de dados. Primeiramente, serão apresentadas as principais técnicas de manipulação de dados utilizando a biblioteca Pandas, vindo a demonstrar na prática operações como seleção de dados, operações de agregamento, agrupamentos e filtros. Em seguida, serão apresentadas as principais bibliotecas de visualização de dados, em especial as bibliotecas Seaborn e Plotly. Serão criados gráficos de diversos tipos com essas bibliotecas, como barra, pizza, linha, boxplot, mapas de calor, dentre outros. Também serão apresentadas as principais técnicas de limpeza e pré-processamento de dados, isto é, trataremos valores nulos, duplicados e outlines. Finalmente, serão discutidos desafios éticos e regulatórios associados ao uso da ciência de dados e análise de dados na saúde.

É importante destacar que este capítulo de livro não aborda a criação de modelos de aprendizado de máquina. Em vez disso, seu foco está nas etapas iniciais da ciência de dados, como manipulação, limpeza e análise exploratória dos dados. Dessa forma, objetivamos desenvolver uma base sólida para futuras aplicações em modelagem preditiva e aprendizado de máquina. Ao explorar esses tópicos, objetivamos contribuir para a capacitação de estudantes, pesquisadores e profissionais interessados em aplicar técnicas de análise de dados para melhorar a eficiência e a qualidade dos serviços médicos. O avanço da ciência de dados na área da saúde tem o potencial de revolucionar a medicina moderna, promovendo diagnósticos mais precisos, tratamentos personalizados e uma gestão hospitalar mais eficiente, sempre com a preocupação de equilibrar inovação tecnológica e responsabilidade ética.

## 3.2. Ciência de Dados na Saúde

A ciência de dados é uma disciplina que tem causado uma profunda transformação em diversos setores da sociedade, e em especial, na área da saúde [Bao et al. 2019]. Esse impacto se deve à capacidade da ciência de dados de extrair conhecimento relevante a partir de grandes volumes de dados, muitas vezes heterogêneos e complexos. Na saúde, o crescimento exponencial na geração de dados biomédicos, clínicos e epidemiológicos nos últimos anos representa uma verdadeira janela de oportunidades. Com a popularização dos prontuários eletrônicos, o uso de dispositivos vestíveis, a digitalização de exames e a expansão dos sistemas públicos de informação em saúde, é possível acessar uma quantidade sem precedentes de informações sobre indivíduos e populações [Teles et al. 2025]. Essa abundância de dados permite a criação de soluções inovadoras baseadas em evidências que podem aprimorar significativamente o diagnóstico precoce de doenças, o monitoramento contínuo de pacientes, a personalização de tratamentos, a gestão eficiente dos recursos hospitalares e o desenho de políticas públicas mais eficazes e equitativas. Assim, a ciência de dados emerge como um eixo estratégico para a modernização do cuidado em saúde, promovendo maior precisão, agilidade e racionalidade nas decisões clínicas, administrativas e governamentais.

A ciência de dados é uma área interdisciplinar que combina conhecimentos oriundos da estatística, aprendizado de máquina, inteligência artificial, domínio específico da área de aplicação (por exemplo, saúde e indústria), dentre outros [Provost and Fawcett 2013]. Seu principal objetivo é extrair conhecimento significativo a partir de grandes volumes de dados, transformando informações brutas em insights úteis para a tomada de decisões estratégicas. Para atingir esse objetivo, a ciência de dados segue um fluxo de trabalho bem definido, composto por diferentes etapas que se complementam.

A Figura 3.1 apresenta as principais etapas de ciência de dados na saúde. Especificamente, o fluxo de trabalho de ciência de dados envolve as seguintes etapas: (i) coleta de dados, que pode envolver desde a extração de registros eletrônicos até o uso de sensores e APIs de sistemas de saúde; (ii) organização e limpeza, que visa padronizar, corrigir erros e tratar valores ausentes ou duplicados; (iii) análise exploratória, responsável por identificar padrões, tendências e possíveis relações entre variáveis; (iv) modelagem e predição, onde técnicas estatísticas e algoritmos são utilizados para construir modelos preditivos ou classificatórios; (v) por fim, a interpretação dos resultados e comunicação, etapa crucial para traduzir os achados técnicos em recomendações práticas.


**Figura 3.1. Fluxo de trabalho típico em ciência de dados na área da saúde.**

Cada uma das etapas apresentadas anteriormente é essencial e interdependente, de modo que a qualidade de uma etapa afeta diretamente o desempenho das demais. Um erro na preparação dos dados, por exemplo, pode comprometer toda a análise subsequente. Por essa razão, compreender a estrutura desse processo e dominar as ferramentas adequadas para cada etapa é fundamental para o sucesso de qualquer projeto de ciência de dados, especialmente em domínios sensíveis e complexos como a saúde.

### 3.2.1. Papel da Ciência de Dados na Melhoria da Saúde Pública

Em sua essência, a saúde é uma área fortemente orientada por dados. Por exemplo, tomadas de decisões referentes a serviços como vigilância epidemiológica e criação de políticas públicas dependem fortemente da coleta, análise e interpretação de grandes volumes de dados [Provost and Fawcett 2013]. Portanto, nesse contexto, a ciência de dados surge como uma solução poderosa para prover compreensões relevantes para apoiar ações preventivas e otimizar alocações de recursos.

A ciência de dados aplicada à saúde possibilita a criação de soluções para monitorar e detectar surtos epidemiológicos em tempo real, reconhecer padrões comportamentais, delinear estratégias para implementação de campanhas de vacinação, otimizar fluxo de trabalho dentro de hospitais, dentre outros serviços [O'connor 2018, Moura et al. 2023]. Um exemplo palpável da aplicabilidade da ciência de dados na saúde foram as tomadas de decisão contra a pandemia de COVID-19 [Latif et al. 2020]. Os gestores e autoridades competentes delinearam ações baseadas em evidências levantadas através de técnicas de ciência de dados, como monitoramento contínuo de dados sobre hospitalizações, óbitos,

vacinação, áreas mais afetadas, dentre outros dados. Especificamente, foram desenvolvidas soluções como painéis interativos contendo diversos tipos de gráficos e modelos de aprendizado de máquina, que representaram ferramentas fundamentais no processo de tomadas de decisão para conter a pandemia de COVID-19.

Além de cenários críticos como a pandemia de COVID-19, a análise de dados desempenha um papel fundamental na vigilância constante de enfermidades crônicas, como transtornos mentais, diabetes, pressão alta e distúrbios cardíacos [Moura et al. 2023], favorecendo a elaboração de estratégias voltadas à prevenção e ao fortalecimento da saúde pública. A integração de informações oriundas de registros clínicos digitais, bancos de dados populacionais, sistemas hospitalares e até tecnologias vestíveis amplia as oportunidades de identificar tendências comportamentais e prever ameaças à saúde coletiva.

Contudo, a aplicação da ciência de dados no âmbito da saúde pública demanda uma abordagem criteriosa, que leve em consideração tanto desafios técnicos quanto implicações éticas. A manipulação de grandes volumes de informações sensíveis exige o compromisso com a proteção da privacidade e da confidencialidade dos dados pessoais, respeitando os direitos individuais dos cidadãos [Arellano et al. 2018]. Portanto, a aplicabilidade da ciência de dados na saúde deve ser aliada à governança de dados e princípios éticos que garantam a equidade em saúde.

### 3.2.2. Aplicações em Saúde: Diagnósticos, Epidemiologia e Tratamentos

A ciência de dados pode ser aplicada em diversos segmentos da saúde, apoiando o processo de tomada de decisão médica em nível clínico e estratégico [Subrahmanya et al. 2022]. Sua aplicabilidade se estende a serviços como auxílio em diagnósticos médicos, monitoramento contínuo de comportamentos, análise epidemiológica, tratamentos personalizados, dentre outros serviços de saúde fundamentados em dados.

Um dos usos mais promissores da ciência de dados é apoiar o diagnóstico e tomada de decisão médica [Teles et al. 2025]. Especificamente, pesquisadores têm analisado e processado dados médicos para implementar modelos de aprendizado de máquina capazes de classificar e predizer estados de saúde. Esses modelos são projetados usando principalmente dados provenientes de fontes como prontuários eletrônicos, imagens médicas e dados de sensores de dispositivos móveis e vestíveis [Teles et al. 2025]. Por exemplo, algoritmos aplicados à análise de imagens podem auxiliar na detecção de câncer, lesões pulmonares e doenças cardiovasculares com alta acurácia [Talwar et al. 2023]. Além disso, sistemas inteligentes têm sido capazes de analisar grandes volumes de dados para detectar padrões comportamentais (por exemplo, padrões de mobilidade, sono e sociabilidade) processando dados de dispositivos pessoais como *smartphone* e *smartwatch* [Moura et al. 2022].

## 3.3. Introdução ao Python

Antes de explorarmos as aplicações do Python na análise de dados, é imprescindível compreender seus conceitos fundamentais. Portanto, nesta seção, apresentaremos os principais conceitos da programação em Python, a saber: (i) indentação em Python; (ii) declaração de variáveis e tipos de dados; (iii) operadores aritméticos, relacionais e booleanos; e (iv) estruturas condicionais. O objetivo desta etapa é fornecer ao leitor uma base sólida para o estudo da análise de dados com Python.

### 3.3.1. Indentação em Python

Uma das características mais marcantes do Python é a indentação obrigatória. Isso significa que os blocos de código são definidos por espaços horizontais (espaços ou tabulações), e não o uso de delimitadores como chaves ({}) ou ponto e vírgula (;), comuns em diversas outras linguagens de programação.

A seguir, apresenta-se um exemplo que ilustra a indentação na linguagem Python. Observe que o recuo do código define, de maneira clara, quais instruções pertencem ao bloco de uma estrutura condicional.

```
1 if True:
2     print("Este bloco está corretamente indentado.")
```

Caso a indentação seja ignorada, o Python gerará um erro. Portanto, respeitar a indentação é essencial para garantir o funcionamento correto do seu programa.

### 3.3.2. Declaração de Variáveis e Tipos de Dados

Variáveis são espaços na memória usados para armazenar dados temporariamente. Em Python, não é necessário declarar o tipo da variável antes de usá-la — basta atribuir um valor com o sinal =. No entanto, é fundamental observar algumas regras para a definição de nomes de variáveis:

- Devem começar com uma letra ou com um underline (_).
- Podem conter letras (a-z, A-Z), números (0-9) e underline.
- Não podem conter espaços, acentos ou caracteres especiais como ($, #, @, !, etc.).
- Não podem usar palavras reservadas do Python, como `print`, `if`, `else`, `True`, `False`, `min`, `max`, etc.

### 3.3.3. Tipos de Dados em Python

A Tabela 3.3.3 apresenta os principais tipos de dados utilizados na linguagem Python, juntamente com suas descrições e exemplos práticos. Cada tipo de dado possui características específicas que determinam como as informações são armazenadas e manipuladas no programa.

A tabela apresenta desde os tipos numéricos, como int (números inteiros) e float (números decimais), até tipos mais complexos, como list e dict. O tipo str é utilizado para armazenar textos, enquanto o tipo bool é empregado para valores lógicos (verdadeiro ou falso). Já as coleções list e tuple permitem armazenar múltiplos valores, com a diferença de que listas são mutáveis (podem ser alteradas após a criação) e tuplas são imutáveis. Por fim, o tipo dict representa um dicionário, estrutura que utiliza pares chave-valor para organizar os dados.

Compreender esses tipos é fundamental para trabalhar de forma eficiente com variáveis e estruturas em Python, além de permitir o desenvolvimento de soluções mais organizadas e eficazes no tratamento de dados.

| **Tipo** | **Descrição** | **Exemplo** |
|---|---|---|
| `int` | Representa números inteiros, positivos ou negativos, sem parte decimal. | `10, -3, 0, 42` |
| `float` | Representa números reais (decimais), permitindo valores fracionários. | `3.14, 0.5, -2.7` |
| `str` | Representa sequências de caracteres, usadas para armazenar textos. | `'Olá', "Python", "123"` |
| `bool` | Representa valores lógicos, indicativos de verdadeiro ou falso. | `True, False` |
| `list` | Coleção ordenada e mutável de elementos, que podem ser de tipos variados. | `[1, 2, 3], ["a", "b", "c"]` |
| `tuple` | Coleção ordenada e imutável de elementos, definida por parênteses. | `(1, 2, 3), ("a", "b")` |
| `dict` | Estrutura de dados composta por pares chave-valor, usada para mapeamentos. | `{"nome": "Ana", "idade": 25}` |

Entender a diferença entre cada um dos tipos de dados e seu funcionamento é essencial para trabalhar com diferentes tipos de datasets e realizar o tratamento correto dos dados.

### 3.3.4. Operadores Aritméticos, Relacionais e Booleanos

Em Python, os operadores matemáticos, relacionais e booleanos são amplamente utilizados ao longo do código. Conhecê-los é essencial para a manipulação de dados, análises e comparações.

Os operadores aritméticos são utilizados para realizar operações matemáticas básicas entre números. A Tabela 3.3.4 apresenta os principais operadores aritméticos utilizados na linguagem Python, acompanhados de suas respectivas descrições e exemplos. Esses operadores são fundamentais para a realização de cálculos e expressões matemáticas nos programas.

| **Operador** | **Descrição** | **Exemplo** |
|---|---|---|
| `+` | Realiza a adição entre dois valores. | `10 + 5 → 15` |
| `-` | Realiza a subtração entre dois valores. | `10 - 3 → 7` |
| `*` | Realiza a multiplicação entre dois valores. | `4 * 2 → 8` |
| `/` | Realiza a divisão entre dois valores. | `10 / 2 → 5.0` |
| `**` | Calcula a exponenciação (potência). | `2 ** 3 → 8` |
| `%` | Retorna o resto da divisão (módulo). | `10 % 3 → 1` |

A Tabela 3.3.4 apresenta de forma concisa algumas das funções matemáticas mais usadas em Python, mostrando o que fazem e como aplicá-las em exemplos práticos.

| **Função** | **O que Faz** | **Exemplo** |
|---|---|---|
| `abs(x)` | Converte qualquer número para sua magnitude positiva. | `abs(-12) → 12` |
| `pow(x, y)` | Eleva `x` à potência `y`; aceita opcionalmente um módulo. | `pow(3, 4) → 81` |
| `sqrt(x)` | Retorna a raiz quadrada de `x`; importe com `from math import sqrt`. | `sqrt(9) → 3.0` |
| `min(iterável)` | Percorre uma coleção e devolve o menor valor; também aceita múltiplos args. | `min([7, 2, 5]) → 2` |
| `max(iterável)` | Retorna o maior elemento de uma sequência ou vários argumentos. | `max([7, 2, 5]) → 7` |

Os operadores relacionais são usados para comparar dois valores e retornam um valor booleano (True ou False), indicando se a comparação é verdadeira ou falsa. A Tabela 3.3.4 apresenta os principais operadores relacionais em Python, suas descrições e exemplos de uso.

| **Operador** | **Descrição** | **Exemplo** |
|---|---|---|
| == | Igual a: verifica se dois valores são iguais. | `5 == 3 → False` |
| != | Diferente de: verifica se dois valores são diferentes. | `5 != 3 → True` |
| > | Maior que: testa se o valor à esquerda é maior que o da direita. | `7 > 2 → True` |
| >= | Maior ou igual a: verifica se um valor é maior ou igual ao outro. | `5 >= 5 → True` |
| < | Menor que: testa se o valor à esquerda é menor que o da direita. | `2 < 7 → True` |
| <= | Menor ou igual a: verifica se um valor é menor ou igual ao outro. | `3 <= 3 → True` |

Os operadores booleanos são usados para realizar operações lógicas entre expressões que retornam valores booleanos (True ou False). A Tabela 3.3.4 apresenta os principais operadores booleanos em Python, suas descrições e exemplos de uso.

| **Operador** | **Descrição** | **Exemplo** |
|---|---|---|
| `and` | Retorna `True` se ambas as condições forem verdadeiras. | `True and False → False` |
| `or` | Retorna `True` se pelo menos uma das condições for verdadeira. | `True or False → True` |
| `not` | Inverte o valor lógico: `True` vira `False` e vice-versa. | `not True → False` |

### 3.3.5. Estruturas Condicionais

Estruturas condicionais são utilizadas para alterar o fluxo do código com base em regras e parâmetros definidos pelo desenvolvedor. Um exemplo clássico de aplicação de estruturas condicionais é o desenvolvimento de um sistema de avaliação de desempenho acadêmico. Suponha que precisamos determinar a situação de um aluno com base na sua nota final, seguindo os critérios abaixo:

- Nota maior ou igual a 7 → Aprovado
- Nota maior ou igual a 4 e menor que 7 → Recuperação
- Nota abaixo de 4 → Reprovado

A implementação dessa lógica em Python pode ser feita utilizando uma estrutura condicional `if-elif-else`, que seleciona o resultado adequado com base na nota fornecida. Veja como isso ficaria no código:

```
notafinal = 5  # Valor de exemplo

# Estrutura de Decisão
if notafinal >= 7:  # SE notafinal for maior ou igual a 7,
   execute isso
    print("O aluno(a) está aprovado!")

elif notafinal >= 4:  # SENÃO, MAS SE notafinal for maior ou
   igual a 4, execute isso
    print("O aluno(a) está de recuperação!")

else:  # SENÃO, execute isso
    print("O aluno(a) está reprovado!")
```

## 3.4. Fundamentos do Python para Análise de Dados

Python é uma linguagem de programação de alto nível, amplamente reconhecida por sua versatilidade, clareza sintática e ampla aplicabilidade em diferentes áreas do conhecimento. Nesta seção, será apresentada a utilização do Python como uma ferramenta fundamental no contexto da análise de dados. Especificamente, iremos abordar os conceitos e fundamentos essenciais que servirão de base para o seu correto uso ao longo deste estudo.

### 3.4.1. Estruturas de Dados: DataFrame e Series no Pandas

O **Pandas** é uma biblioteca de código-aberto para Python que fornece ferramentas de alto desempenho e fáceis de usar para manipulação e análise de dados. Seu principal atrativo é a capacidade de trabalhar de forma eficiente com dados tabulares (planilhas, bases relacionais, arquivos CSV, etc.), permitindo filtrar, agrupar, transformar e resumir conjuntos de dados de forma muito mais simples do que usando apenas listas e dicionários.

**Series** são estruturas unidimensionais do Pandas: vetores rotulados capazes de armazenar qualquer tipo de dado (numérico, texto, data, etc.). Cada elemento de uma Series possui um índice (label), o que facilita:

- Seleção de faixas e alinhamento automático em operações aritméticas;
- Integração com outras bibliotecas (NumPy, Matplotlib etc.).

**DataFrame** é a estrutura bidimensional do Pandas, equivalente a uma tabela de banco de dados ou a uma planilha de Excel. Internamente, um DataFrame é composto por várias

Series compartilhando o mesmo índice de linhas, mas com colunas independentes que podem ter tipos diferentes. Isso o torna ideal para:

- *Leitura e escrita* de formatos diversos (CSV, Excel, JSON, SQL);
- *Seleção e filtragem* de linhas e colunas de forma intuitiva;
- *Transformações vetorizadas*, aplicando funções a colunas inteiras em uma única operação;
- *Agrupamentos e agregações* para resumos estatísticos (soma, média, contagem, etc.);
- *Junções e mesclagens* de tabelas distintas, como em bancos relacionais.

### 3.4.2. Preparando o Ambiente

Antes de iniciar a análise de dados, é fundamental garantir que o ambiente esteja devidamente configurado. Além do Pandas utilizaremos bibliotecas como, NumPy, Seaborn e Plotly. Recomendamos o uso do Google Colab (`https://colab.research.google.com/`) ou Jupyter Notebook (`https://jupyter.org/`), pois essas plataformas facilitam a visualização de resultados e gráficos interativos.

Se estiver utilizando um ambiente local, você pode instalar as bibliotecas com o seguinte comando:

```
!pip install pandas numpy seaborn plotly
```

No Google Colab, essas bibliotecas geralmente já estão instaladas por padrão. Neste trecho de código abaixo, realizamos a importação das bibliotecas que serão utilizadas ao longo de todo o conteúdo.

```
import pandas as pd
import numpy as np
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
```

### 3.4.3. Importação de Dados para Análise

Nesta seção, utilizaremos um dataset sobre diabetes, que contém informações sobre pacientes e diversos fatores que podem influenciar o risco de desenvolver a doença. Para fazer o download do arquivo `.csv`, acesse o link para download do CSV.

O objetivo é aplicar técnicas de análise de dados para explorar, entender e extrair insights valiosos a partir dessas informações. O dataset inclui variáveis como: número de gestações, níveis de glicose, pressão arterial, espessura da pele, insulina, IMC, histórico familiar de diabetes, idade e o resultado do teste de diabetes.

Agora, vamos carregar o conjunto de dados que utilizaremos ao longo das próximas seções. Abaixo, apresentamos o código para ler diretamente o CSV do Google Drive usando o método `read_csv` do Pandas:

```
document_id = "1kH99hEJLsT1B2d5DWUUN2i1ATRdiy9Uj"
url = f"https://drive.google.com/uc?id={document_id}&export=
    download"

diabetes_df = pd.read_csv(url)
```

Se preferir, você também pode baixar o arquivo pelo link acima e enviá-lo manualmente ao Google Colab (aba "Files" → "Upload"). Em seguida, basta carregá-lo com:

```
import pandas as pd

diabetes_df = pd.read_csv("diabetes.csv")
```

### 3.4.4. Manipulação de Dados

A manipulação de dados é uma etapa essencial na análise. É nela que organizamos e preparamos os dados para que estejam prontos para serem analisados, incluindo ações como renomear colunas, criar novas variáveis e visualizar estatísticas básicas.

Para facilitar a leitura e interpretação do conjunto de dados, é recomendável renomear e padronizar os nomes das colunas, tornando-os mais claros e descritivos. No exemplo a seguir, realizamos essa padronização atribuindo novos nomes diretamente ao atributo columns do DataFrame.

```
diabetes_df.columns = [
    'Gravidez',
    'Glicotese',
    'Pressao',
    'Pele',
    'Insulina',
    'IMC',
    'Hereditariedade',
    'Idade',
    'Diabetes'
]
```

Dessa forma, os nomes originais são substituídos por termos em português, que representam de forma clara as informações contidas em cada coluna do dataset.

#### 3.4.4.1. Explorando Estatísticas Descritivas

Uma das primeiras etapas na análise exploratória de dados é obter uma visão geral das principais características numéricas do conjunto de dados. O Pandas oferece o método

describe(), que fornece estatísticas como média, desvio padrão, valores mínimos e máximos para cada coluna numérica. É essencial para obter uma visão geral dos dados e identificar possíveis valores extremos. O código abaixo apresenta seu uso no DataFrame diabetes_df e a Figura 3.2 apresenta o resultado da execução desse código.

```
diabetes_df.describe()
```

| | Gravidez | Glicotese | Pressao | Pele | Insulina | IMC | Hereditariedade | Idade | Diabetes |
|---|---|---|---|---|---|---|---|---|---|
| count | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| mean | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | 0.471876 | 33.240885 | 0.348958 |
| std | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 | 11.760232 | 0.476951 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 | 21.000000 | 0.000000 |
| 25% | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 | 24.000000 | 0.000000 |
| 50% | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| 75% | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| max | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

**Figura 3.2. Resultado da Função `Describe`.**

O método describe() gera um resumo estatístico automático das colunas numéricas que incluem as seguintes informações:

- count: quantidade de valores não nulos;
- mean: média aritmética;
- std: desvio padrão;
- min: valor mínimo;
- 25%, 50% e 75%: quartis (percentis que indicam a distribuição dos dados);
- max: valor máximo.

### 3.4.5. Selecionando Linhas e Colunas

Uma das tarefas fundamentais na análise de dados é a seleção de informações específicas dentro de um DataFrame. O Pandas oferece maneiras simples e flexíveis de acessar tanto colunas quanto linhas, permitindo que o analista foque apenas nos dados relevantes para cada etapa da análise.

Nas primeiras etapas, é comum realizar uma inspeção inicial para compreender sua estrutura e verificar se a leitura foi realizada corretamente. Para isso, é possível exibir rapidamente o início e o fim do DataFrame usando os métodos ‘head()‘ e ‘tail()‘, conforme ilustrado nas Figuras 3.3 e 3.4.

```
# Exibe as 5 primeiras linhas (padrão)
diabetes_df.head()
```

| | Gravidez | Glicotese | Pressao | Pele | Insulina | IMC | Hereditariedade | Idade | Diabetes |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

**Figura 3.3. Resultado da função `Head`.**

```
# Exibe as 10 últimas linhas (definido pelo parâmetro)
diabetes_df.tail(10)
```

| | Gravidez | Glicotese | Pressao | Pele | Insulina | IMC | Hereditariedade | Idade | Diabetes |
|---|---|---|---|---|---|---|---|---|---|
| 758 | 1 | 106 | 76 | 0 | 0 | 37.5 | 0.197 | 26 | 0 |
| 759 | 6 | 190 | 92 | 0 | 0 | 35.5 | 0.278 | 66 | 1 |
| 760 | 2 | 88 | 58 | 26 | 16 | 28.4 | 0.766 | 22 | 0 |
| 761 | 9 | 170 | 74 | 31 | 0 | 44.0 | 0.403 | 43 | 1 |
| 762 | 9 | 89 | 62 | 0 | 0 | 22.5 | 0.142 | 33 | 0 |
| 763 | 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 | 0 |
| 764 | 2 | 122 | 70 | 27 | 0 | 36.8 | 0.340 | 27 | 0 |
| 765 | 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 | 0 |
| 766 | 1 | 126 | 60 | 0 | 0 | 30.1 | 0.349 | 47 | 1 |
| 767 | 1 | 93 | 70 | 31 | 0 | 30.4 | 0.315 | 23 | 0 |

**Figura 3.4. Resultado da função `Tail`.**

Observação: Tanto o head() quanto o tail() permitem a definição da quantidade de linhas que se deseja visualizar, passando um número inteiro como argumento.

Além de exibir as primeiras e últimas linhas de um DataFrame, também é possível acessar linhas de qualquer intervalo, o que é bastante útil para examinar partes específicas dos dados sem precisar percorrer todo o conjunto. No exemplo a seguir, utilizamos a notação de fatiamento do Pandas para exibir as linhas que estão no intervalo de índices 20 a 24 (Figura 3.5).

```
diabetes_df[20:25]
```

| | Gravidez | Glicotese | Pressao | Pele | Insulina | IMC | Hereditariedade | Idade | Diabetes |
|---|---|---|---|---|---|---|---|---|---|
| **20** | 3 | 126 | 88 | 41 | 235 | 39.3 | 0.704 | 27 | 0 |
| **21** | 8 | 99 | 84 | 0 | 0 | 35.4 | 0.388 | 50 | 0 |
| **22** | 7 | 196 | 90 | 0 | 0 | 39.8 | 0.451 | 41 | 1 |
| **23** | 9 | 119 | 80 | 35 | 0 | 29.0 | 0.263 | 29 | 1 |
| **24** | 11 | 143 | 94 | 33 | 146 | 36.6 | 0.254 | 51 | 1 |

**Figura 3.5. Resultado do intervalo `[20:25]`.**

Essa notação remete a uma lista, onde o primeiro valor representa o índice inicial (inclusive) e o segundo valor indica o índice final (exclusive). Ou seja, no Python, os intervalos sempre param no número anterior ao final especificado. Portanto, o código acima mostrará as linhas de índice 20 a 24.

Após a inspeção inicial dos dados, é comum surgir a necessidade de realizar seleções mais específicas, como acessar colunas, linhas ou até mesmo valores individuais dentro do DataFrame. Para esse propósito, o Pandas disponibiliza dois métodos principais de seleção:

- .loc[]: permite acessar dados com base nos rótulos do índice.
- .iloc[]: realiza a seleção com base na posição numérica, onde tanto linhas quanto colunas são indexadas a partir de zero.

O exemplo abaixo ilustra o uso dos métodos .loc[] e .iloc[] para acessar linhas específicas de um DataFrame:

```
# Usando loc para acessar uma linha específica por índice
diabetes_df.loc[10]

# Usando iloc para acessar uma linha pela posição
diabetes_df.iloc[10]

# Intervalo de linhas com iloc (da linha 5 até a 9)
diabetes_df.iloc[5:10]

# Acessando um valor específico (linha 0, coluna 'Idade')
df.loc[0, 'Idade']
```

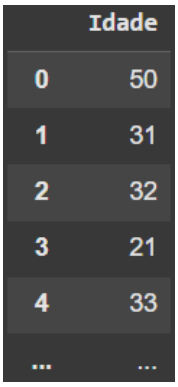
Essas ferramentas são essenciais quando queremos inspecionar, editar ou filtrar os dados de forma precisa.

#### 3.4.5.1. Selecionando Colunas

Além de selecionar linhas, também é possível acessar colunas de forma simples, utilizando os nomes definidos no DataFrame. A Figura 3.6 ilustra a seleção da coluna 'Idade' do DataFrame 'diabetes_df'.

```
diabetes_df['Idade']
```



| | Idade |
|---|---|
| 0 | 50 |
| 1 | 31 |
| 2 | 32 |
| 3 | 21 |
| 4 | 33 |
| ... | ... |

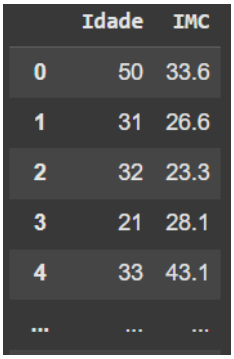
**Figura 3.6. Selecionando a coluna `idade`.**

Explicação: Basta informar o nome do DataFrame (diabetes_df) seguido por colchetes com o nome da coluna desejada entre aspas simples ou duplas.

Para selecionar duas ou mais colunas simultaneamente, basta passar uma lista com os nomes das colunas desejadas dentro dos colchetes:

```
diabetes_df[['Idade', 'IMC']]
```

Dessa forma, retornamos apenas as colunas especificadas, o que é útil para análises focadas em variáveis específicas, como mostrado na Figura 3.7.



| | Idade | IMC |
|---|---|---|
| 0 | 50 | 33.6 |
| 1 | 31 | 26.6 |
| 2 | 32 | 23.3 |
| 3 | 21 | 28.1 |
| 4 | 33 | 43.1 |
| ... | ... | ... |

**Figura 3.7. Selecionando as colunas `idade e IMC`.**

### 3.4.6. Operações de Agregamento e Agrupamento

Ao trabalhar com conjuntos de dados, muitas vezes é necessário resumir informações ou identificar padrões a partir de agrupamentos. O Pandas oferece recursos poderosos para realizar operações de agregação, como soma, média, contagem, valor máximo e mínimo, entre outras.

#### 3.4.6.1. Agregações Básicas

Utilizando funções de agregação do Pandas, é possível obter rapidamente dados relevantes sobre o DataFrame analisado. O Pandas oferece diversas funções de agregação que permitem calcular dados estatísticos essenciais sobre um conjunto de dados. Entre as funções mais comuns estão:

- `count()`: retorna o número de valores não nulos em cada coluna ou série.
- `min()`: retorna o valor mínimo observado em cada coluna numérica.
- `max()`: retorna o valor máximo observado em cada coluna numérica.
- `mean()`: calcula a média aritmética dos valores de cada coluna numérica.
- `std()`: calcula o desvio-padrão dos valores de cada coluna numérica.

O código a seguir ilustra o uso de funções de agregação para realizar cálculos sobre diferentes colunas do 'diabetes_df'. Além disso, a Figura 3.8 apresenta o resultados da execução desses códigos.

```
# Média da idade
diabetes_df['Idade'].mean()

# Maior valor de IMC
diabetes_df['IMC'].max()

# Menor valor de Glicotese
diabetes_df['Glicotese'].min()

# Contagem de entradas na coluna 'Diabetes'
diabetes_df['Diabetes'].count()

# Retorna o desvio padrão dos valores da coluna
diabetes_df['Insulina'].std()
```

| | |
|---|---|
| Média da idade | 33,240885416666664 |
| Maior valor de IMC | 67,1 |
| Menor valor de Glicotese | 0 |
| Contagem de ntradas na coluna 'Diabetes' | 768 |
| Desvio padrão da Insulina | 115,24408931535337 |

**Figura 3.8. Resultado das funções de agregação no DataFrame diabetes.**

### 3.4.6.2. Estatísticas Descritivas Agrupadas por Categorias

O Pandas fornece o método groupby() para segmentar dados em grupos com base em uma ou mais colunas. Esse método permite realizar operações específicas sobre cada grupo, facilitando a análise de dados de maneira estruturada e eficiente. Ao aplicar o groupby(), os dados são divididos em subconjuntos, e funções agregadoras podem ser aplicadas a cada grupo individualmente.

Abaixo, apresentamos um código que utiliza o método groupby() para realizar análises segmentadas com base na coluna 'Diabetes' do conjunto de dados 'diabetes_df'. A seguir, aplicamos a função de agregação 'mean()' para calcular o valor médio para cada grupo. Enfatizamos que poderíamos ter utilizado qualquer outra função de agregação em combinação com o método groupby(), como max(), min(), std(), dentre outros. A Figura 3.9 apresenta o resultado da execução desse código.

```
# Média de idade para cada grupo (diabéticos e não-diabéticos)
diabetes_df.groupby('Diabetes')['Idade'].mean()
```

| Diabetes | Idade |
|---|---|
| 0 | 31.190000 |
| 1 | 37.067164 |

dtype: float64

**Figura 3.9. Resultado do agrupamento pelas colunas Diabetes e Idade.**

No exemplo de código abaixo, utilizamos o método groupby() combinado com a função agg() para calcular várias métricas de agregação para diferentes colunas do DataFrame, agrupando os dados com base na coluna 'Diabetes' (diabéticos e não-diabéticos). A Figura 3.10 apresenta o resultado da execução do código abaixo.

```
# Várias métricas por grupo
diabetes_df.groupby('Diabetes').agg({
    'Idade': ['mean', 'max'],
    'IMC': ['mean', 'min']
}).reset_index() # A função reset_index serve para ajustar a
   visualização.
```

| Diabetes | Idade | | IMC | |
|---|---|---|---|---|
| | mean | max | mean | min |
| 0 | 31.190000 | 81 | 30.304200 | 0.0 |
| 1 | 37.067164 | 70 | 35.142537 | 0.0 |

**Figura 3.10. Resultado da função `Group`.**

#### 3.4.6.3. Contagem de Ocorrências Únicas

O pandas fornece o método value_counts() para contar a frequência de valores únicos em uma coluna de um DataFrame. Esse método realiza a contagem de cada valor único presente, ordenada de forma decrescente (do valor mais frequente para o menos frequente). Por exemplo, no código abaixo, esse método foi utilizado para contar a frequência de cada valor único na coluna 'Gravidez' do DataFrame 'diabetes_df'. A Figura 3.11 apresenta o retorno da execução desse código. Especificamente, é retornada uma Série com os valores únicos encontrados na coluna 'Gravidez' e suas respectivas contagens, ordenadas do valor mais frequente para o menos frequente.

```
diabetes_df['Gravidez'].value_counts()
```

| Gravidez | |
|---|---|
| 1 | 135 |
| 0 | 111 |
| 2 | 103 |
| 3 | 75 |
| 4 | 68 |
| 5 | 57 |

**Figura 3.11. Resultado da função `value_counts`.**

### 3.4.7. Filtros

Filtrar dados significa criar subconjuntos que atendam a uma condição lógica. Isso é útil, por exemplo, para estudar apenas pacientes com IMC elevado ou apenas aqueles acima

de certa idade. O código abaixo apresenta exemplos de filtros de dados. O primeiro filtra pacientes com idades superiores a 30 anos. O segundo código filtra pacientes que possuem mais de trinta anos e que apresentam IMC acima de 25. Por fim, o último código filtra pacientes com diabetes ou que apresentem IMC acima de 35.

```
# Pacientes com mais de 30 anos
diabetes_df[diabetes_df['Idade'] > 30]

# Pacientes com mais de 30 anos E IMC acima de 25
diabetes_df[(diabetes_df['Idade'] > 30) & (diabetes_df['IMC'] >
    25)]

# Pacientes com diabetes OU IMC acima de 35
diabetes_df[(diabetes_df['Diabetes'] == 1) | (diabetes_df['IMC']
     > 35)]
```

Lembre-se de sempre utilizar parênteses em torno das condições ao usar os operadores lógicos & (E) e | (OU).

### 3.4.8. Ordenação

Para ordenar os dados em ordem crescente ou decrescente, utilizamos o método sort_values(). Esse método permite organizar os dados com base em uma ou mais colunas, por meio do parâmetro by. Já o parâmetro ascending define se a ordenação será crescente (True) ou decrescente (False). No código abaixo, apresentamos alguns exemplos de ordenação de dados. O primeiro exemplo, diabetes_df.sort_values('Idade') ordena os dados pela coluna 'Idade' em ordem crescente. Já diabetes_df.sort_values('IMC', ascending=False) organiza os dados da coluna 'IMC' do maior para o menor valor. Também é possível ordenar por múltiplos critérios, como em diabetes_df.sort_values(by=['Diabetes', 'Idade'], ascending=[False, True]), que ordena primeiro pela coluna 'Diabetes' em ordem decrescente e, em seguida, pela coluna 'Idade' em ordem crescente.

```
# Ordenar por idade (crescente)
diabetes_df.sort_values('Idade')

# Ordenar por IMC (decrescente)
diabetes_df.sort_values('IMC', ascending=False)

# Ordenar por múltiplos critérios
diabetes_df.sort_values(by=['Diabetes', 'Idade'], ascending=[
    False, True])
```

Ordenar os dados é uma prática comum para facilitar análises, gerar gráficos ou destacar os extremos.

## 3.5. Limpeza e Tratamentos de Dados

A limpeza e tratamento de dados é uma das etapas cruciais no processo de análise de dados. Independentemente da fonte ou formato, raramente os dados chegam em condições

ideais para análise. Dados incompletos, inconsistentes, duplicados ou fora do padrão são desafios comuns enfrentados por cientistas de dados. Nessa fase, o objetivo é garantir que o conjunto de dados esteja confiável, coerente e pronto para ser explorado com técnicas analíticas e modelos preditivos. Portanto, nesta seção, objetivamos realizar a limpeza e o tratamento de dados utilizando as ferramentas do *python*, com foco na biblioteca Pandas.

### 3.5.1. Configuração do Ambiente

A execução das etapas de limpeza e tratamento de dados demonstradas nessa seção usarão as bibliotecas Pandas, NumPy e Seaborn. Como apresentado nas seções anteriores, essas ferramentas são amplamente exploradas na análise de dados e oferecem funcionalidades robustas para manipulação, inspeção e visualização de dados. Portanto, abaixo apresentamos o código de importação necessário para usar essas bibliotecas.

```
import pandas as pd
import numpy as np
import seaborn as sns
```

A seguir, apresentamos o código que cria um DataFrame simulado com dados de saúde, representando um conjunto de informações coletadas em um estudo clínico fictício com pacientes. O objetivo é reproduzir, de forma prática, situações comuns encontradas na preparação de dados reais, como: (i) presença de valores ausentes (por falhas na coleta ou entrada de dados); (ii) registros duplicados (por exemplo, quando um paciente é registrado mais de uma vez); (iii) valores discrepantes (outliers), que podem ser causados por erros de digitação ou por dados extremos fora da faixa esperada. A opção por dados simulados permite controlar intencionalmente a presença de valores ausentes, duplicados e discrepantes, garantindo que cada tipo de problema possa ser demonstrado e tratado de forma clara e objetiva.

```
dados_saude = pd.DataFrame({
    'id': [
        101, 102, 103, 104, 105, 106, 106, 107
    ],
    'nome': [
        'Ana', 'Bruno', 'Carlos', 'Diana', 'José',
        'Fernando', 'Fernando', 'Gabriela'
    ],
    'idade': [
        29, 35, 42, np.nan, 150, 38, 38, 27
    ],
    'peso_kg': [
        65.0, 80.5, np.nan, 70.0, 60.0,
        300.0, 300.0, 48.0
    ],
    'pressao_sistolica': [
        120, 130, 125, 118,
        119, 250, 250, np.nan
    ],
    'fumante': [
```

```
        'Não', 'Sim', 'Não', 'Não', 'Sim',
        np.nan, np.nan, 'Não'
    ],
    'data_coleta': [
        2024-01-10', '2024-01-12', '2024-01-15',
        '2024-01-17', '2024-01-19', '2024-01-21',
        '2024-01-21', '2024-01-23'
    ]
})
```

Ao executar o código anterior, resultará na criação do DataFrame denominado 'dados_saude' apresentado na Figura 3.12, que contém as seguintes colunas: 'id', 'nome', 'idade', 'peso_kg', 'pressao_sistolica', 'fumante' e 'data_coleta'. Esse DataFrame já é suficiente para trabalhar os seguintes pontos didáticos: valores ausentes (np.nan em idade, peso_kg, pressao_sistolica, fumante), tratamento de dados duplicados (id e nome duplicados no caso de Fernando); valores discrepantes (idade = 150, peso_kg = 300, pressao_sistolica = 250) e conversão de tipo de dados.

| | id | nome | idade | peso_kg | pressao_sistolica | fumante | data_coleta |
|---|---|---|---|---|---|---|---|
| 0 | 101 | Ana | 29.0 | 65.0 | 120.0 | Não | 2024-01-10 |
| 1 | 102 | Bruno | 35.0 | 80.5 | 130.0 | Sim | 2024-01-12 |
| 2 | 103 | Carlos | 42.0 | NaN | 125.0 | Não | 2024-01-15 |
| 3 | 104 | Diana | NaN | 70.0 | 118.0 | Não | 2024-01-17 |
| 4 | 105 | Eduarda | 150.0 | 60.0 | 119.0 | Sim | 2024-01-19 |
| 5 | 106 | Fernando | 38.0 | 300.0 | 250.0 | NaN | 2024-01-21 |
| 6 | 106 | Fernando | 38.0 | 300.0 | 250.0 | NaN | 2024-01-21 |
| 7 | 107 | Gabriela | 27.0 | 48.0 | NaN | Não | 2024-01-23 |

**Figura 3.12. Dataframe com dados simulados de saúde contendo valores ausentes, duplicados e discrepantes.**

### 3.5.2. Análise Inicial do DataFrame

Antes de iniciar qualquer processo de limpeza e tratamento de dados, é fundamental entender a estrutura básica do conjunto de dados que estamos manipulando. Uma das primeiras ferramentas que o pandas oferece para essa tarefa é o método **.info()**. A Figura 3.13 apresenta o resultado ao executar o comando 'dados_saude.info()'. A partir desse resultado, é possível identificar as seguintes informações: (i) o DataFrame possui 8 linhas e 7 colunas; (ii) existem valores ausentes nas colunas idade, peso_kg, pressao_sistolica, pois a quantidade de valores não nulos é menor que a quantidade total de linhas; e (iii) as colunas estão com tipos incorretos de dados (por exemplo, idade está float e data_coleta está object).

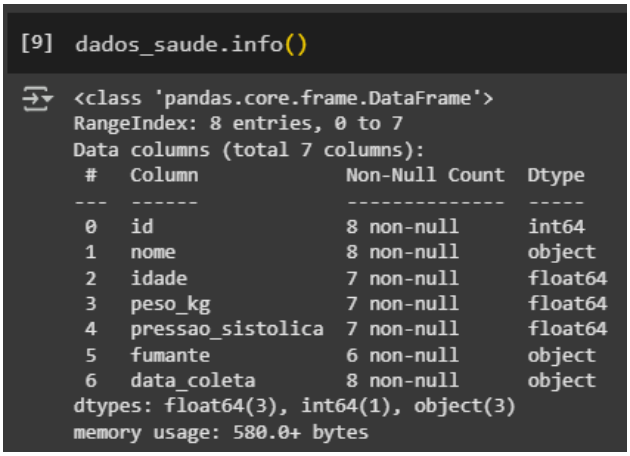



**Figura 3.13. Resultado da execução do comando .info() aplicado ao DataFrame dados_saude.**

Neste momento temos uma visão global de nosso conjunto de dados e, identificamos problemas que devem ser tratados, como valores ausentes, e colunas com tipos incorretos. Nas próximas seções, apresentaremos um conjunto de técnicas de higienização de dados que serão aplicadas a esse DataFrame.

### 3.5.3. Tratamento de Valores Ausentes

Em dados do mundo real, é comum encontrar informações ausentes em conjuntos de dados, que são conhecidas como valores nulos ou NaN (Not a Number). Tratar esses valores corretamente é uma tarefa fundamental de higienização de dados, já que eles podem atrapalhar análises, distorcer gráficos e comprometer resultados de modelos de aprendizado de máquina.

O primeiro passo para lidar com valores ausentes é identificar quais colunas contêm essas lacunas. Para isso, o Pandas oferece o método **.isna()**, que verifica cada posição do DataFrame e retorna uma tabela com valores booleanos: True onde há dados ausentes e False onde os dados estão preenchidos. Como o resultado mantém o mesmo formato do DataFrame original, fica fácil visualizar onde estão as falhas. Além disso, combinando esse método com **.sum()**, é possível contar o total de valores ausentes em cada coluna. Abaixo apresentamos o código que realiza a contagem de valores ausentes no DataFrame dados_saude. A Figura 3.14 apresenta o resultado do comando que realiza a contagem de valores ausentes. Ao analisar o resultado, observamos que há exatamente um valor ausente nas colunas nome, idade e peso_kg. Já a coluna fumante apresenta dois valores ausentes.

```
1 dados_saude.isna().sum()
```


**Figura 3.14. Resultado da contagem de valores ausentes no DataFrame dados_saude.**

Após identificar os valores ausentes, é necessário preenchê-los com um valor específico, como a média, mediana, zero ou uma string padrão. Com esse objetivo, o pandas fornece o método **.filna()**, que é capaz de preencher as lacunas vazias por dados de nosso interesse. O código abaixo trata os valores ausentes de dados_saude usando o método **.fillna()**. Primeiramente, preenchemos o valor ausente da coluna idade com a mediana ( método **.median()** ) das idades. Em seguida, preenchemos o valor faltante da coluna peso_kg com a média dos pesos ( **.mean()** ). Para a coluna pressao_sistolica usamos a estratégia de substituir pelo valor zero. Por fim, na coluna fumante, preenchemos os valores ausentes com a mensagem 'Não informado'.

```
idade_median = dados_saude['idade'].median()
dados_saude['idade'] = dados_saude['idade'].fillna(idade_median)

peso_mean = dados_saude['peso_kg'].mean()
dados_saude['peso_kg'] = dados_saude['peso_kg'].fillna(peso_mean
    )

dados_saude['pressao_sistolica'] = dados_saude['
    pressao_sistolica'].fillna(0)

dados_saude['fumante'] = dados_saude['fumante'].fillna('Não
    informado')
```

Em algumas situações, é necessário realizar a remoção das linhas que contêm valores ausentes, em vez de preenchê-los. Para este objetivo, o pandas fornece o método **.dropna()**, que elimina todas as linhas que contenham valores ausentes. Além disso, o método .dropna() permite o uso do parâmetro inplace=True. Quando esse parâmetro é definido como verdadeiro, a remoção das linhas ocorre diretamente no DataFrame original, sem a necessidade de criar uma nova variável ou sobrescrever manualmente. Caso o inplace não seja especificado ou seja definido como False, o método apenas retorna uma nova versão do DataFrame, deixando o original inalterado. Portanto, em nosso exemplo, caso a melhor decisão de higienização de dados fosse remover todas as linhas com valores ausentes, poderíamos executar o código abaixo.

```
dados_saude.dropna(inplace=True)
```

Neste momento, se executarmos novamente o comando **dados_saude.isna().sum()**, verificaremos que não existem mais valores ausentes, uma vez que tratamos usando o método **.fillna()** e o método **.dropna()**.

### 3.5.4. Remoção de Duplicatas

Outro problema comum que encontramos em conjuntos de dados é a duplicação de registros, que também pode distorcer as análises e visualizações de dados geradas e, portanto, precisa ser tratada cuidadosamente. O primeiro passo para tratar desse problema é identificar se o nosso DataFrame possui dados duplicados. Para isso, o pandas fornece o método .duplicated(), que é utilizado para identificar linhas repetidas dentro de um DataFrame. Ele retorna uma lista de valores booleanos, em que cada linha é marcada como True se for uma cópia exata de alguma linha anterior (considerando todas as colunas, por padrão). O código abaixo usa esse comando para verificar linhas duplicadas no DataFrame dados_saude. A Figura 3.15 apresenta o resultado da execução desse código, no qual é possível identificar que a linha 6 representa um registro duplicado, pois retornou um valor booleano verdadeiro.

```
dados_saude.duplicated()
```


**Figura 3.15. Resultado da identificação de linhas duplicadas no DataFrame dados_saude.**

O próximo passo que iremos executar é a remoção dessa linha duplicada. O pandas fornece o método **.drop_duplicates()** que é capaz de remover todos os registros duplicados de um DataFrame, mantendo apenas uma ocorrência (por padrão, a primeira). Além disso, esse método também permite o uso do parâmetro inplace=True, para que a remoção das duplicatas seja feita diretamente no DataFrame original, sem a necessidade de atribuir o resultado a uma nova variável. O código abaixo remove as linhas duplicadas do DataFrame dados_saude.

```
dados_saude.drop_duplicates()
```

Neste momento, se executarmos novamente o comando **dados_saude.duplicated()**, verificaremos que não existem mais registros duplicados, uma vez que tratamos esse problema usando o método **.drop_duplicates()**.

### 3.5.5. Conversão de tipos

Em tarefas de ciência de dados, é comum que os dados importados de diferentes fontes apresentem tipos inadequados para as análises. Por exemplo, no DataFrame dados_saude, a coluna idade e pressao_sistolica foram carregadas com o tipo float, quando o apropriado seria o tipo int. Esse tipo de inconsistência pode comprometer tanto as análises estatísticas quanto as visualizações de dados.

Para solucionar esse problema, o Pandas disponibiliza o método **.astype()**, que permite converter o tipo de dados de uma ou mais colunas de forma simples e eficiente. No exemplo a seguir, utilizamos esse método para converter as colunas idade e pressao_sistolica para o tipo inteiro (int), e as colunas nome e fumante para o tipo string.

```
dados_saude['idade'] = dados_saude['idade'].astype(int)
dados_saude['pressa_sistolica'] = dados_saude['pressao_sistolica
    '].astype(int)
dados_saude['nome'] = dados_saude['nome'].astype('string')
dados_saude['fumante'] = dados_saude['fumante'].astype('string')
```

Além das colunas tratadas anteriormente, também é possível identificar que a coluna 'data_coleta' está com o tipo incorreto. Essa inconsistência pode dificultar operações usuais em campos temporais, como calcular diferenças de tempo, ordenar cronologicamente e extrair componentes da data (como ano, mês e dia). Com o propósito de tratar esse problema, o pandas oferece o método .to_datetime(), que converte o tipo de colunas para representarem datas em objetos do tipo datetime, permitindo que essas informações sejam interpretadas corretamente pelo Python. Abaixo, apresentamos o código que utiliza esse método para converter o tipo da coluna 'data_coleta' para date.

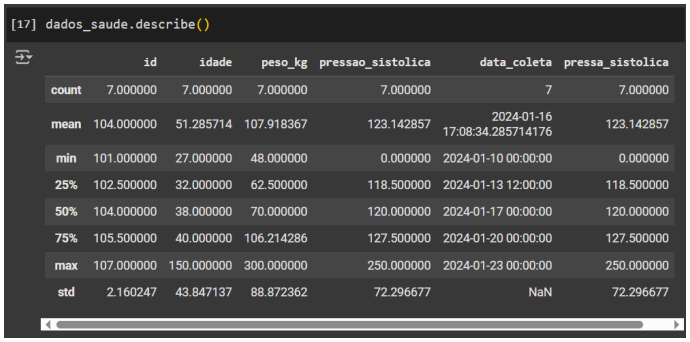
```
dados_saude['data_coleta'] = pd.to_datetime(dados_saude['
    data_coleta'])
```

### 3.5.6. Lidando com Outliers

Em ciência de dados, um passo importante é identificar e tratar valores discrepantes (isto é, outliers) presentes no conjunto de dados. Esses outliers podem ser resultado de diversos fatores, como erros de digitação e falhas de medição de sensores e equipamentos. No DataFrame 'dados_saude', é possível identificar alguns casos de outliers, a saber: (i) um paciente com idade de 150 anos; (ii) um peso corporal registrado como 300 kg; e (iii) uma pressão sistólica de 250, que está acima dos limites fisiológicos humanos comuns.

A primeira forma de identificar valores discrepantes é usar o método .describe() (apresentado na Seção 3.4.4.1), que resume as informações numéricas das colunas e permite perceber valores que não se ajustam ao intervalo esperado. A Figura 3.16 detalha as informações estatísticas do DataFrame 'dados_saude', no qual é possível identificar valores fora da faixa esperada para algumas colunas.



```
[17] dados_saude.describe()
```

| | id | idade | peso_kg | pressao_sistolica | data_coleta | pressa_sistolica |
|---|---|---|---|---|---|---|
| count | 7.000000 | 7.000000 | 7.000000 | 7.000000 | 7 | 7.000000 |
| mean | 104.000000 | 51.285714 | 107.918367 | 123.142857 | 2024-01-16 17:08:34.285714176 | 123.142857 |
| min | 101.000000 | 27.000000 | 48.000000 | 0.000000 | 2024-01-10 00:00:00 | 0.000000 |
| 25% | 102.500000 | 32.000000 | 62.500000 | 118.500000 | 2024-01-13 12:00:00 | 118.500000 |
| 50% | 104.000000 | 38.000000 | 70.000000 | 120.000000 | 2024-01-17 00:00:00 | 120.000000 |
| 75% | 105.500000 | 40.000000 | 106.214286 | 127.500000 | 2024-01-20 00:00:00 | 127.500000 |
| max | 107.000000 | 150.000000 | 300.000000 | 250.000000 | 2024-01-23 00:00:00 | 250.000000 |
| std | 2.160247 | 43.847137 | 88.872362 | 72.296677 | NaN | 72.296677 |

**Figura 3.16. Resultado da execução do método .describe() no DataFrame dados_saude.**

Outra maneira prática de detectar outliers é utilizando visualizações gráficas, como o boxplot. Essa ferramenta gráfica é bastante eficiente para identificar valores extremos, pois resume a distribuição dos dados de forma visual e intuitiva. Esse tipo de gráfico será detalhado na Seção 3.6.

Após identificar os outliers, é necessário aplicar alguma estratégia para tratá-los. A primeira estratégia que pode ser aplicada é remover todas as linhas que contêm valores discrepantes. Outra alternativa viável é substituir os outliers por valores plausíveis, usando estatísticas como média e mediana. Além disso, também é possível aplicar regras de negócio previamente estabelecidas, definindo limites máximos e mínimos aceitáveis com base no conhecimento especializado da área e, assim, filtrar os dados que estiverem fora desses parâmetros.

No caso do Dataframe dados_saude, trataremos outliers usando a estratégia de remoção de linhas com valores discrepantes. Especificamente, utilizaremos filtros (conforme apresentado na Seção 3.4.7) para remover outliers. Abaixo, apresentamos o código que trata os valores discrepantes em dados_saude.

```
# Removendo idade acima de 120
dados_saude = dados_saude[dados_saude['idade'] <= 120]

# Removendo peso acima de 200 kg
dados_saude = dados_saude[dados_saude['peso_kg'] <= 200]

# Removendo pressão sistólica acima de 200
dados_saude = dados_saude[dados_saude['pressao_sistolica'] <=
    200]
```

### 3.5.7. Salvando o Dataset Tratado

Após realizar todo o processo de limpeza de dados, é necessário salvar o dataset tratado para garantir que as alterações realizadas sejam preservadas e possam ser utilizadas em análises futuras. Essa prática evita a repetição desnecessária do processo de limpeza sempre que os dados forem utilizados novamente, além de garantir a consistência e a reprodutibilidade das análises. O pandas oferecem métodos simples que permitem salvar DatFrames em diversos formatos de arquivos, como JSON, CSV, Excel, dentre outros formatos. No código abaixo,

utilizamos o método '.to_csv()' para salvar o DataFrame dados_saude em arquivo com formato CSV. Além disso, atribuímos valor falso ao parâmetro index, para que os índices sejam ignorados ao criar o arquivo.

```
dados_saude.to_csv('dados_saude_tratados.csv', index=False)
```

Neste momento, finalizamos todas as etapas de limpeza e tratamento de dados em nosso DataFrame dados_saude. Utilize os conhecimentos adquiridos nessa seção para verificar se o conjunto de dados foi realmente higienizado (por exemplo, use os métodos .info e .describe). Em conclusão, dominar técnicas de limpeza de dados é essencial para qualquer profissional da área, já que dados bem preparados são a base para conclusões mais precisas e modelos mais robustos.

## 3.6. Explorando Dados com Gráficos na Saúde

A exploração visual de dados é uma das etapas mais importantes da ciência de dados, pois permite identificar padrões, tendências, distribuições e possíveis anomalias de maneira visual e intuitiva. Esta importância é ressaltada quando lidamos com dados da área da saúde, pois visualizações gráficas podem ser uma ferramenta poderosa em tarefas como tomadas de decisões clínicas, políticas públicas e apoio a diagnósticos médicos.

Nesta seção, exploraremos o uso das bibliotecas Seaborn e Plotly para criar gráficos a partir de um conjunto de dados sobre pacientes com e sem histórico de Acidente Vascular Cerebral (AVC). Os gráficos serão empregados para ilustrar relações entre variáveis, distribuições de valores, proporções e agrupamentos relevantes. Através dessa abordagem, seremos capazes de compreender melhor este conjunto de dados e extrair insights relevantes para o contexto médico.

### 3.6.1. Configuração do Ambiente

Antes de começarmos a explorar os dados por meio de visualizações, é necessário preparar o ambiente de trabalho com as bibliotecas que serão utilizadas ao longo desta seção. Abaixo, apresentamos o código que importa as bibliotecas pandas, matplotlib, seaborn e plotly.

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import plotly.express as px
```

Com as bibliotecas devidamente configuradas, podemos seguir para o carregamento dos dados. O código abaixo faz a leitura do conjunto de dados sobre pacientes com e sem histórico de AVC diretamente do Google Drive. Caso opte por baixar os dados e carregar manualmente, o leitor pode baixar o conjunto de dados a partir deste link: `https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset`.

```
dataset_code = '1DP5u0SzZXtgb7mfs07cXjzrUwOE0rDEd'
url = f'https://drive.google.com/uc?id={dataset_code}'
```

```
avc_df = pd.read_csv(url)
```

Com os dados devidamente carregados, podemos visualizar as colunas e as cinco primeiras linhas do DataFrame 'avc_df' executando o código abaixo. A Figura 3.17 apresenta o resultado da execução desse código. Analisando esse resultado, é possível identificar que o DataFrame avc_df reúne informações clínicas e demográficas de pacientes para análise de fatores associados ao AVC. Esse conjunto de dados inclui colunas como id (identificador do paciente), gender (gênero), age (idade), hypertension e heart_disease (indicam presença dessas condições), ever_married (histórico de casamento), work_type (tipo de ocupação), Residence_type (zona de residência), avg_glucose_level (nível médio de glicose), bmi (índice de massa corporal), smoking_status (histórico de tabagismo) e stroke (indica se o paciente sofreu AVC).

```
avc_df.head()
```

| | id | gender | age | hypertension | heart_disease | ever_married | work_type | Residence_type | avg_glucose_level | bmi | smoking_status | stroke | faixa_idade |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 9046 | Male | 67.0 | 0 | 1 | Yes | Private | Urban | 228.69 | 36.6 | formerly smoked | 1 | 51–70 |
| 1 | 51676 | Female | 61.0 | 0 | 0 | Yes | Self-employed | Rural | 202.21 | NaN | never smoked | 1 | 51–70 |
| 2 | 31112 | Male | 80.0 | 0 | 1 | Yes | Private | Rural | 105.92 | 32.5 | never smoked | 1 | 71+ |
| 3 | 60182 | Female | 49.0 | 0 | 0 | Yes | Private | Urban | 171.23 | 34.4 | smokes | 1 | 31–50 |
| 4 | 1665 | Female | 79.0 | 1 | 0 | Yes | Self-employed | Rural | 174.12 | 24.0 | never smoked | 1 | 71+ |

**Figura 3.17. Detalhamento do DataFrame avc_df que contém dados sobre pacientes com e sem histórico de AVC.**

No decorrer desta seção, iremos utilizar as bibliotecas gráficas Seaborn e Plotly para criar diversos tipos de gráficos usando esse conjunto de dados. Enfatizamos também que utilizaremos os nomes de colunas originais dos dados (isto é, em inglês), mas, caso necessário, o leitor poderá mudar para português conforme ensinado na Seção 3.4.4.

### 3.6.2. Criação de Gráficos com a Biblioteca Seaborn

A biblioteca Seaborn é uma poderosa ferramenta de visualização de dados baseada no Matplotlib, projetada para tornar a criação de gráficos estatísticos mais simples e informativa. Nesta seção, exploraremos como utilizar o Seaborn para gerar visualizações no contexto da saúde, a partir do nosso DataFrame avc_df.

#### 3.6.2.1. Histograma: Distribuição de Idade dos Pacientes

Histograma é um tipo de gráfico de barras utilizado para representar a distribuição de uma variável numérica. Ele agrupa os dados em intervalos (chamados de bins) e mostra quantos valores caem em cada intervalo. No código a seguir, utilizamos a Seaborn para criar um histograma da variável age do conjunto de dados avc_df, que representará a distribuição de idades dos pacientes. O primeiro comando sns.histplot(avc_df['age'], bins=10, kde=True, color='red') gera o gráfico com 10 intervalos de idade (isto é, 10 bins), adiciona uma linha de densidade para suavizar a visualização da distribuição (kde) e define a cor das barras como vermelha. Em seguida, configuramos o título do gráfico e os rótulos dos eixos

utilizando plt.title, plt.xlabel e plt.ylabel. Por fim, exibimos o gráfico com o comando plt.show(). A Figura 3.18 apresenta o gráfico gerado, no qual é possível observar como as idades dos pacientes estão distribuídas.

```
#Plota o histograma da idade dos pacientes
sns.histplot(avc_df['age'], bins=10, kde=True, color='red')
#Insere um título para o gráfico
plt.title('Distribuição de Idade dos Pacientes')
#Adicionam rótulo ao eixo X
plt.xlabel('Idade')
#Adicionam rótulo ao eixo Y
plt.ylabel('Frequência')
#Exibe o gráfico
plt.show()
```

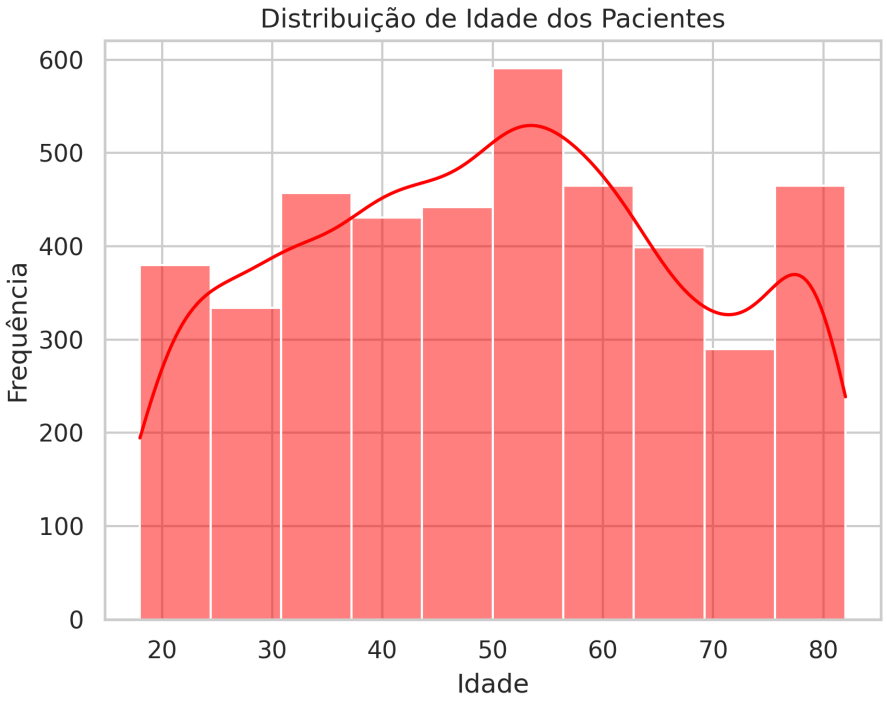



**Figura 3.18. Histograma que exibe a frequência de faixas etárias.**

### 3.6.2.2. Gráfico de Contagem: Relação Hipertensão x AVC

O gráfico de contagem (ou count plot) é utilizado para visualizar a frequência de categorias em uma variável categórica, permitindo comparações rápidas entre os grupos. No código abaixo, utilizamos a Seaborn para criar um gráfico de contagem relacionando a presença de hipertensão (hypertension) com a ocorrência de AVC (stroke). O primeiro comando sns.countplot() constrói o gráfico considerando hypertension no eixo x e separando as contagens pelo valor de stroke (usando cores distintas definidas pela paleta 'viridis').

Configuramos o título e os rótulos dos eixos com plt.title, plt.xlabel e plt.ylabel, e exibimos o gráfico com o comando plt.show().

```
sns.countplot(x='hypertension', hue='stroke', data=avc_df,
   palette='viridis')
plt.title('Relação entre Hipertensão e Ocorrência de AVC')
plt.xlabel('Hipertensão (0 = Não, 1 = Sim)')
plt.ylabel('Quantidade')
plt.show()
```

A Figura 3.19 apresenta o gráfico gerado, que possibilita observar se pacientes hipertensos tiveram uma frequência maior ou menor de AVC em comparação aos pacientes sem hipertensão.

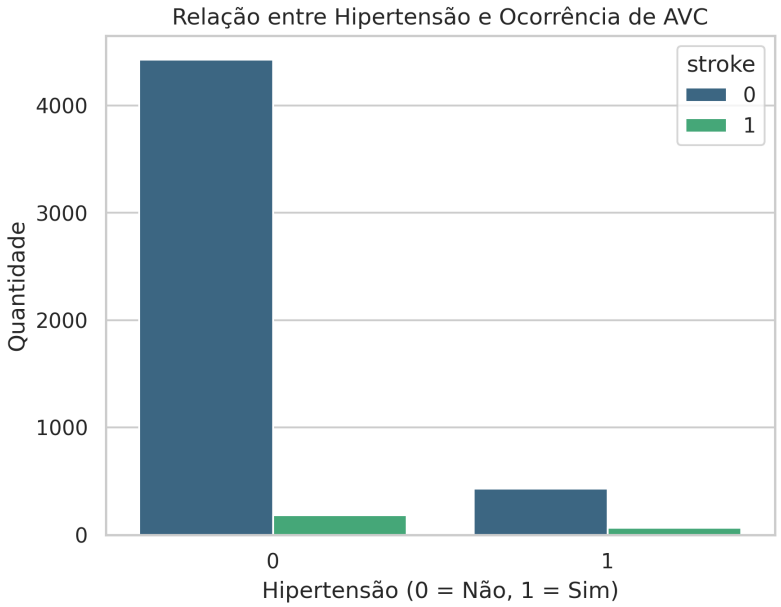



**Figura 3.19. Gráfico countplot que representa a relação entre hipertensão e AVC.**

### 3.6.2.3. Gráfico de Dispersão: Relação Idade x Glicose

O gráfico de dispersão (scatterplot) é utilizado para identificar a relação entre duas variáveis numéricas. Esse gráfico plota um ponto baseado nas coordenadas de cada valor, isto é, sua posição é determinada pelos valores das duas variáveis escolhidas, uma no eixo X e outra no eixo Y. No código abaixo, criamos um gráfico de dispersão usando o comando sns.scatterplot() para identificar a relação entre a idade dos pacientes e seu nível médio de glicose, distinguindo-os conforme a presença ou ausência de hipertensão. No eixo X, representamos a idade (age) e no eixo Y, o nível médio de glicose (avg_glucose_level). As cores dos pontos indicam se o paciente é hipertenso (1) ou não (0), utilizando a paleta de cores coolwarm. A legenda foi configurada para deixar claro o significado dos valores de hipertensão (isto é, 0 = não, 1 = sim).

```
sns.scatterplot(x='age', y='avg_glucose_level', data=avc_df, hue
    ='hypertension', palette='coolwarm')
plt.title('Idade vs Nível Médio de Glicose - Diferenciando
    Hipertensão')
plt.xlabel('Idade (anos)')
plt.ylabel('Nível Médio de Glicose')
plt.legend(title='Hipertensão (0 = Não, 1 = Sim)')
plt.show()
```

A Figura 3.20 apresenta o gráfico gerado, que possibilita identificar se há alguma tendência, como níveis mais elevados de glicose em pacientes hipertensos de determinadas faixas etárias. Especificamente, identificamos que pacientes com idades mais avançadas (isto é, acima de 50 anos) tendem a sofrer de hipertensão e apresentar altos níveis de glicose.

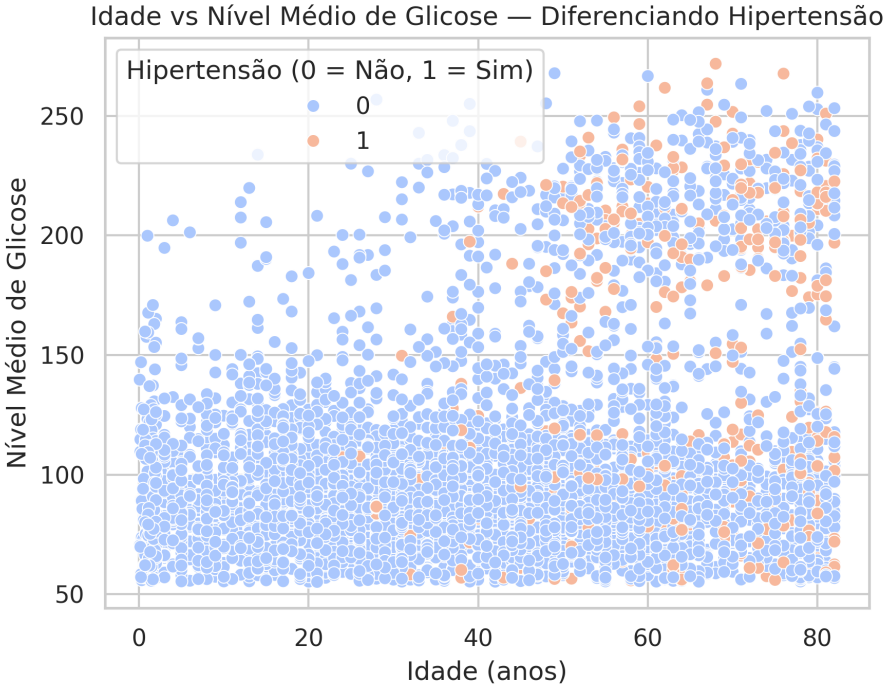



**Figura 3.20. Gráfico de dispersão que representa a relação entre idade e nível médio de glicose.**

### 3.6.2.4. Box Plot: BMI por Tipo de Trabalho

O boxplot é um gráfico utilizado para mostrar a distribuição de um conjunto de dados numéricos, destacando seus principais componentes estatísticos. Ele fornece uma maneira eficaz de visualizar a mediana, a variabilidade e os outliers de uma variável. Os principais elementos de um boxplot são:

- Caixas: representam o intervalo interquartil, que vai do primeiro quartil (Q1) ao

terceiro quartil (Q3). A caixa contém 50% dos dados, sendo que a linha dentro da caixa indica a mediana (Q2).

- Bigodes: as linhas que se estendem da caixa mostram os valores máximos e mínimos dentro de 1.5 vezes o intervalo interquartil a partir dos quartis. Valores fora desse intervalo são considerados outliers.

- Outliers: são os pontos que ficam fora da área definida pelos bigodes, podendo indicar dados incomuns ou discrepantes.

O código abaixo cria um boxplot que visualiza a distribuição do Índice de Massa Corporal (bmi) para diferentes tipos de trabalho (work_type). Para tanto, utilizamos a função sns.boxplot() do Seaborn, com a variável work_type no eixo X e bmi no eixo Y. O argumento hue='work_type' permite colorir as caixas de acordo com o tipo de trabalho, facilitando a distinção visual entre as categorias. A paleta de cores utilizada é pastel, proporcionando cores suaves. O comando plt.title() define o título do gráfico e os comandos plt.xlabel() e plt.ylabel() são usados para rotular os eixos X e Y, respectivamente. O comando plt.xticks(rotation=45) inclina os rótulos no eixo X, melhorando assim a legibilidade do gráfico.

```
sns.boxplot(x='work_type', y='bmi', data=avc_df, hue='work_type'
    , palette='pastel')
plt.title('Distribuição do IMC por Tipo de Trabalho')
plt.xlabel('Tipo de Trabalho')
plt.ylabel('Índice de Massa Corporal (IMC)')
plt.xticks(rotation=45)
plt.show()
```

A Figura 3.21 apresenta o gráfico gerado, que possibilita observar diferenças entre as categorias e identificar outliers. Especificamente, identificamos que o imc dos pacientes que nunca trabalharam (never_worked) é significativamente menor em comparação com os demais tipos de trabalho. Além disso, reconhecemos que existe uma maior quantidade de outliers nos dados dos pacientes que trabalham no setor privado.

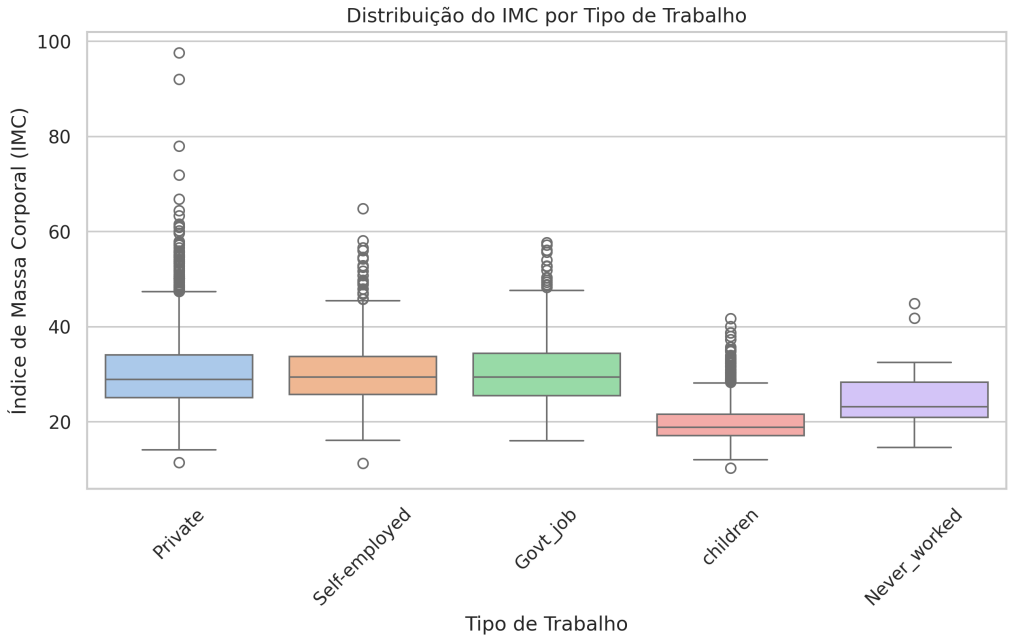



**Figura 3.21. Box plot que representa a distribuição dos dados e revela possíveis outliers de bmi por tipo de trabalho.**

### 3.6.2.5. Mapa de Calor: Correlações Entre Variáveis Numéricas

Mapa de calor (heatmap) é um gráfico que representa valores por meio de tonalidades de cores, em vez de apenas números. Esse tipo de gráfico possui uma escala de tonalidades que facilita a identificação de padrões e correlações entre variáveis. Seu uso mais comum é criar visualizações de matrizes de correlação, já que ele destaca rapidamente quais variáveis têm relações mais fortes (positivas ou negativas) entre si. O código abaixo cria um mapa de calor a partir das correlações das variáveis numéricas do DataFrame avc_df. Primeiramente, calculamos a matriz de correlação usando o comando avc_df.corr(numeric_only=True), que resulta em valores entre -1 (correlação fortemente negativa) e 1 (correlação fortemente positiva). Em seguida, usamos o comando sns.heatmap() para criar o mapa de calor com os seguintes parâmetros: annot=True que define que os valores numéricos sejam exibidos, cmap='coolwarm' define uma paleta de cores que varia do azul (correlação negativa) ao vermelho (correlação positiva), e fmt=".2f"que formata os números para duas casas decimais.

```
corr = avc_df.corr(numeric_only=True)
sns.heatmap(data=corr,annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlações')
plt.show()
```

A Figura 3.22 apresenta o mapa de calor gerado, que apresenta a força da relação entre as variáveis numéricas. Especificamente, as cores com tons vermelhos fortes indicam correlações fortes (positivas ou negativas), enquanto tons azuis fortes indicam pouca ou nenhuma correlação.

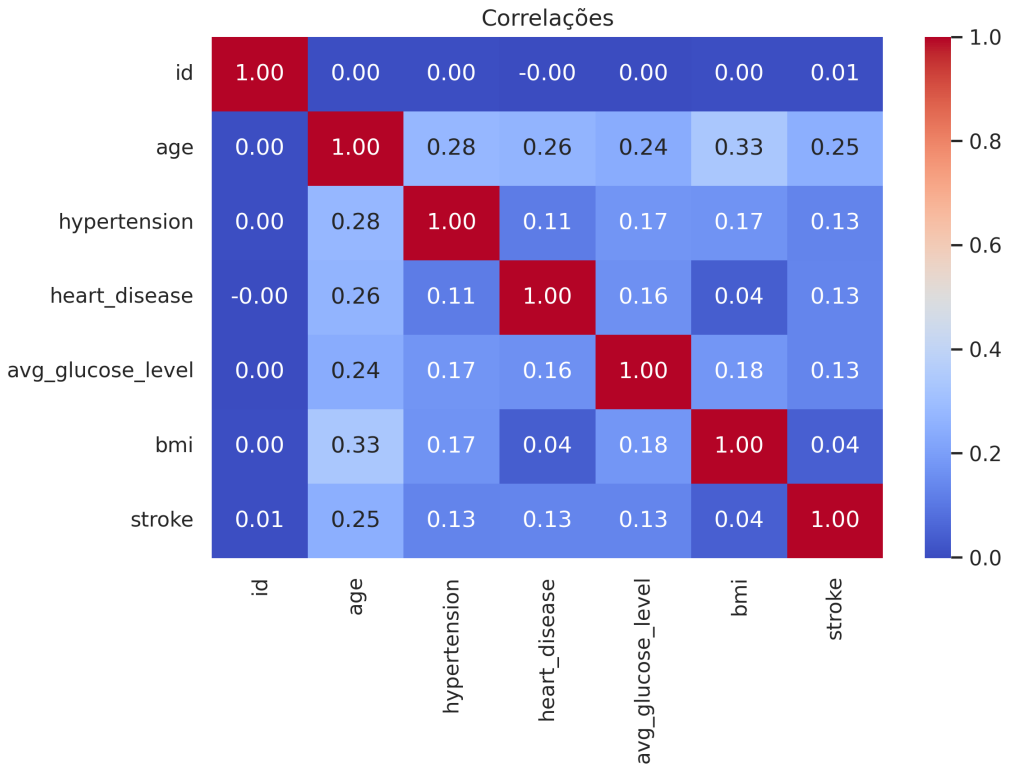



**Figura 3.22. Mapa de calor das correlações entre as variáveis do DataFrame avc_df.**

### 3.6.3. Criação de Gráficos com a Biblioteca Plotly

Nesta seção, aprenderemos a criar gráficos interativos usando a biblioteca Plotly. Essa biblioteca permite criar gráficos interativos e dinâmicos de dados, o que é especialmente útil em análises exploratórias e apresentações. Embora as imagens apresentadas nesta seção sejam gráficos estáticos, é importante ressaltar que todos os gráficos criados com Plotly são interativos. Isso significa que, ao visualizar o gráfico em um ambiente compatível (por exemplo, Google Colab), você pode passar o mouse sobre os elementos para ver informações detalhadas, fazer zoom e mover-se pelo gráfico. Portanto, aprenderemos a construir diversos tipos de gráficos com Plotly utilizando o conjunto de dados avc_df. Através de exemplos práticos, mostraremos como representar informações de forma clara e atrativa, favorecendo a análise e a tomada de decisões.

#### 3.6.3.1. Gráfico de Pizza Interativo: Proporção de Incidência de AVC por Tipo de Trabalho

Um gráfico de pizza é uma visualização circular usada para mostrar a proporção de diferentes categorias dentro de um todo. Cada fatia representa uma categoria, e seu tamanho é proporcional à quantidade ou frequência dessa categoria. O código abaixo cria um gráfico de pizza utilizando a biblioteca Plotly Express (px) para representar a proporção de casos de AVC conforme o tipo de trabalho dos pacientes. O comando px.pie() é utilizado para gerar o gráfico, onde apresenta os seguintes parâmetros: names='work_type' define que cada fatia representará uma categoria desta coluna; values='stroke' indica que o

tamanho das fatias será proporcional ao número de casos de AVC registrados; e title define o título do gráfico. Em seguida, fig.update_traces(textinfo='percent+label') ajusta as informações exibidas em cada fatia, mostrando tanto o nome da categoria quanto a porcentagem que ela representa em relação ao total. Por fim, o comando fig.show() exibe o gráfico interativo na tela.

```
fig = px.pie(avc_df,
             names='work_type',
             values='stroke',
             title='Proporção de AVCs por Tipo de Trabalho'
)

fig.update_traces(textinfo='percent+label')
fig.show()
```

A Figura 3.23 apresenta o gráfico de pizza gerado, em que cada fatia representa a proporção de pacientes com AVC dentro de uma categoria de tipo de trabalho.

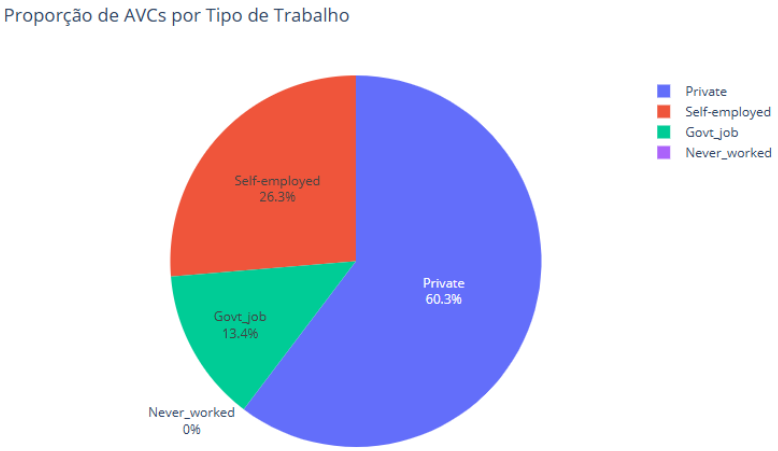



**Figura 3.23. Gráfico de pizza que apresenta a proporção de incidência de AVC por tipo de Trabalho.**

### 3.6.3.2. Gráfico de Violino: Distribuição do Nível Médio de Glicose por Estado Civil

Um gráfico de violino (violin plot) é uma visualização que combina o boxplot e a densidade de probabilidade. Especificamente, este gráfico adiciona a visualização da concentração de dados ao boxplot, em que a forma de 'violino' reflete a densidade dos dados. O código abaixo cria um gráfico de violino para visualizar a distribuição do nível médio de glicose entre diferentes grupos de estado civil. Utilizamos o comando px.violin() para criar esse gráfico, em que foram usados os seguintes parâmetros: x='ever_married' e y='avg_glucose_level' para definir o eixo x e y, respetivamente; color='ever_married' para diferenciar por cores os dois grupos de estado civil (casados e não casados); box=True para adicionar um boxplot dentro do gráfico de violino, fornecendo informações sobre a mediana e os quartis da distribuição; points='all' para exibir todos os pontos de dados no

gráfico; e definimos o parâmetro ‘title’ para plotar o título do gráfico. Por fim, definimos os rótulos dos eixos X e Y usando o comando fig.update_layout() para melhorar a legibilidade do gráfico.

```
fig = px.violin(avc_df,
                x='ever_married',  y='avg_glucose_level', color=
                   'ever_married', box=True,
                points='all',
                title='Distribuição do Nível Médio de Glicose
                   por Estado Civil')
fig.update_layout(
    xaxis_title='Estado Civil',
    yaxis_title='Nível Médio de Glicose'
)
fig.show()
```

A Figura 3.24 apresenta o gráfico de violino gerado, que permite observar como os níveis de glicose variam entre os grupos de estado civil, mostrando a distribuição, mediana e dispersão dos dados. No ambiente de execução, o leitor poderá interagir com o gráfico, executando ações como zoom e passar o mouse por cima para visualizar informações com mais detalhes.

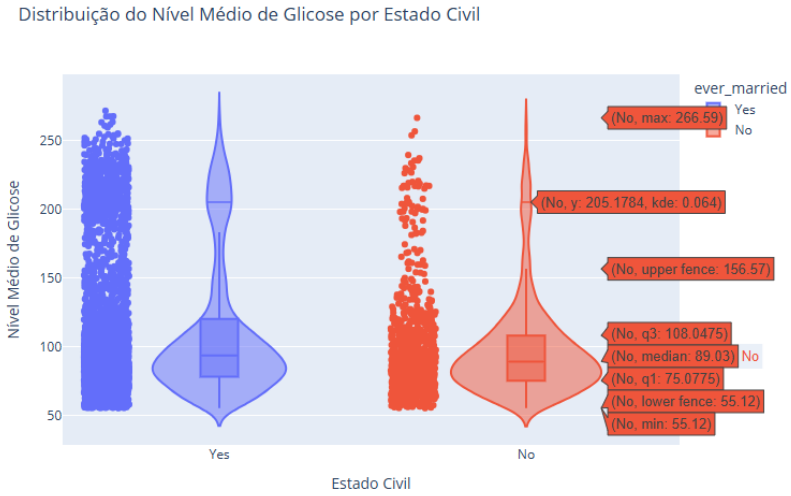



**Figura 3.24. Gráfico de violino da distribuição do nível médio de glicose por estado civil.**

### 3.6.3.3. Gráfico Sunburst: Distribuição Hierárquica entre Gênero Trabalho e Hipertensão

O sunburst é um tipo de gráfico hierárquico em formato circular, onde cada nível é representado por um anel. Ele apresenta como as categorias se subdividem e qual a proporção de cada segmento em relação ao todo. No código abaixo, utilizamos o método px.sunburst() para criar um gráfico do tipo sunburst, que representa hierarquicamente a relação entre gênero, tipo de trabalho e hipertensão. Primeiramente, definimos cada nível

da hierarquia usando o parâmetro path=['gender', 'work_type', 'hypertension'], isto é, o centro representa o gênero, o próximo nível o tipo de trabalho e, por fim, a presença ou não de hipertensão. A coloração dos segmentos é baseada na coluna stroke, que indica se o paciente teve ou não AVC. O método fig.update_traces(textinfo='label+percent entry') adiciona ao gráfico os rótulos das categorias e a porcentagem correspondente em relação ao total de entradas. Por fim, fig.show() exibe o gráfico.

```
fig = px.sunburst(
    avc_df,
    path=['gender', 'work_type', 'hypertension'],
     color='stroke',
    title='Distribuição Hierárquica por Gênero, Tipo de Trabalho
        e Hipertensão'
)
# Mostrar os rótulos nas seções
fig.update_traces(textinfo='label+percent entry')
fig.show()
```

A Figura 3.25 apresenta o gráfico sunburst gerado, que apresenta a distribuição hierárquica dos pacientes por gênero, tipo de trabalho e presença de hipertensão, com cores indicando a ocorrência de AVC. É possível visualizar como essas categorias se subdividem e contribuem proporcionalmente para o total de registros.

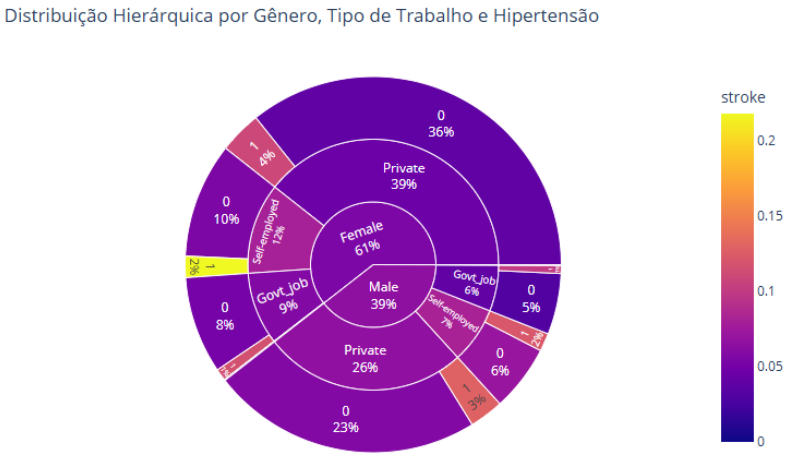



**Figura 3.25. Gráfico sunburst que apresenta de forma hierárquica a relação entre gênero, tipo de trabalho e hipertensão.**

### 3.6.3.4. Gráfico de Barras: Proporção de Hipertensão por Faixas de Idade

Nesta seção, iremos criar um gráfico de barras para apresentar a proporção de pacientes com hipertensão agrupados por faixas de idade. Antes da criação deste gráfico, vamos realizar um processamento inicial dos dados para permitir a análise por faixas etárias. O código abaixo apresenta esse processamento. Especificamente, realizamos as seguintes etapas: Primeiramente, criamos a coluna age_range utilizando o método pd.cut(), que segmenta a variável contínua age em quatro faixas: Até 30 anos, 31–50 anos, 51–70 anos e

71+ anos. Em seguida, os dados são agrupados com base nessas faixas etárias, e calculamos a média da variável hypertension em cada grupo. Como essa variável é binária (0 para ausência e 1 para presença de hipertensão), a média representa diretamente a proporção de indivíduos com hipertensão em cada faixa. Por fim, essa proporção é convertida para porcentagem, facilitando a interpretação no gráfico que será gerado.

```
# Criação da coluna faixa_idade com pd.cut()
avc_df['age_range'] = pd.cut(avc_df['age'],bins=[0, 30, 50, 70,
    100],
                             labels=['Até 30 anos', '31-50 anos',
                                     '51-70 anos', '71+ anos'])

# Agrupamento e cálculo da proporção de AVC por faixa etária
hipertencao_df = avc_df.groupby('age_range').agg(
    hypertension_proportion = ('hypertension', 'mean')
).reset_index()

# Convertendo a proporção para um formato percentual
hipertencao_df['hypertension_percent'] =
                    hipertencao_df['hypertension_proportion'] *
                       100
```

Após o processamento inicial, podemos utilizar o DataFrame hipertencao_df para criar o gráfico de barras. O código a seguir gera uma visualização da proporção de hipertensão em diferentes faixas etárias. A função px.bar() é utilizada para construir o gráfico, onde o eixo X representa as faixas etárias (age_range) e o eixo Y exibe a proporção percentual de pacientes com hipertensão (hypertension_percent). As cores das barras são definidas com base nas faixas etárias, facilitando a distinção visual entre os grupos. Em seguida, o método fig.update_layout() é empregado para configurar os rótulos dos eixos e o título da legenda.

```
fig = px.bar(hipertencao_df,
        x='age_range', y='hypertension_percent', color='
           age_range',
        title='Proporção de Hipertensão por Faixa Etária')

fig.update_layout(
    xaxis_title='Faixa Etária',
    yaxis_title='Proporção de Hipertensão',
    legend_title_text='Faixa Etária'
)
fig.show()
```

A Figura 3.26 apresenta o resultado: um gráfico de barras interativo que ilustra a proporção de pacientes com hipertensão em cada faixa etária, com base nos dados previamente agrupados e convertidos em percentuais.

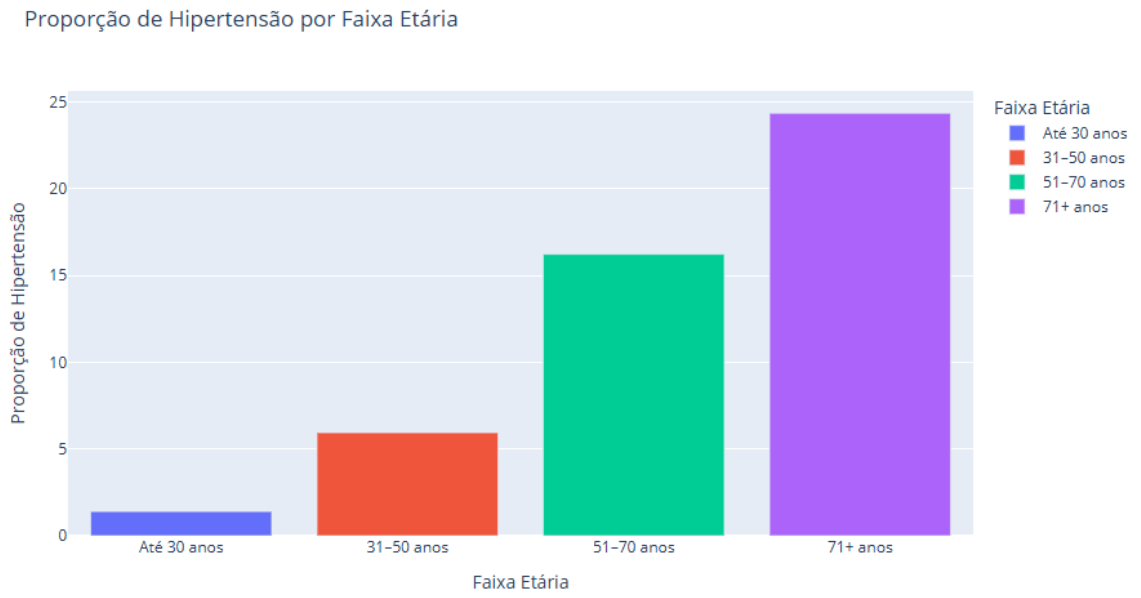



**Figura 3.26. Gráfico de barras que apresenta a proporção de hipertensão por faixas de idade.**

### 3.6.3.5. Gráfico de Linha: Glicose Média por Idade

Um gráfico de linhas (line plot) é um tipo de visualização que mostra a variação de um ou mais valores ao longo de um eixo contínuo, geralmente o tempo. Ele conecta os pontos de dados com linhas, facilitando a identificação de padrões e comparações entre séries. O código abaixo cria um gráfico de linhas que apresenta a variação da glicose média em função da idade dos pacientes. Antes de criar o gráfico efetivamente, realizamos dois processamentos iniciais, a saber: (i) aplicamos um filtro para considerar apenas pacientes com 18 anos ou mais e (ii) agrupamos os dados por idade e calculamos a média dos níveis de glicose para cada idade. O resultado é armazenado no DataFrame media_glicose_df. No próximo passo, utilizamos o método px.line() para criar um gráfico de linhas com marcadores (markers=True), em que o eixo X representa a idade e o eixo Y mostra a glicose média correspondente. Por fim, usamos o método fig.update_layout() para definir os rótulos dos eixos e o gráfico é exibido com a execução do fig.show().

```
#Filtro de idade
avc_df = avc_df[avc_df['age'] >= 18]

# Cálculo da glicose média por idade
media_glicose_df = avc_df.groupby('age').agg(
    media_glicose = ('avg_glucose_level', 'mean')
).reset_index()

fig = px.line(media_glicose_df,
              x='age',
              y='media_glicose',
              markers=True,
```

```
                title='Glicose Média por Idade')

fig.update_layout(
    xaxis_title='Idade',
    yaxis_title='Glicose Média',
)
fig.show()
```

A Figura 3.27 apresenta a variação da glicose média em função da idade dos pacientes. O eixo X representa as idades, enquanto o eixo Y exibe a glicose média correspondente a cada faixa etária. As linhas conectam os pontos de dados, facilitando a visualização das tendências. Cada ponto é marcado, destacando a glicose média para cada idade. Esse gráfico ajuda a entender como os níveis médios de glicose variam à medida que a idade dos pacientes aumenta.

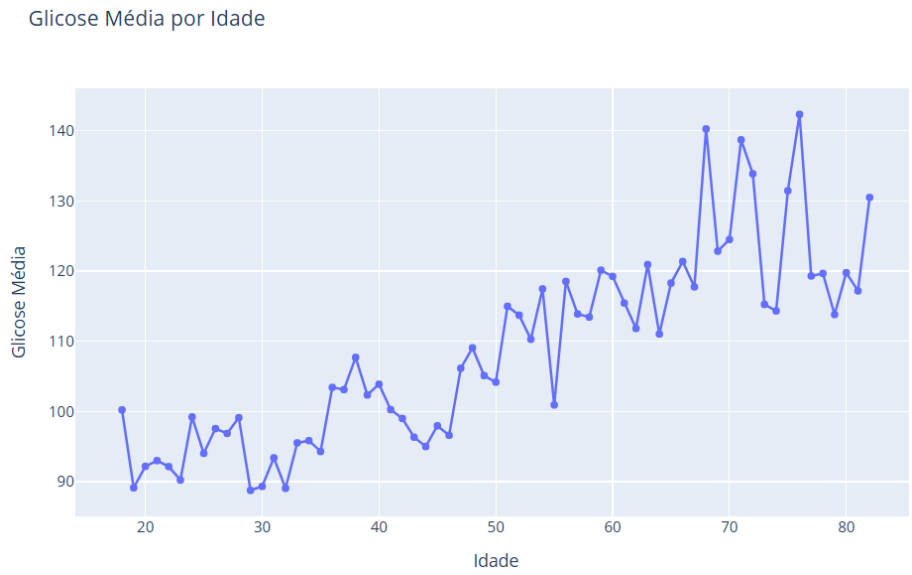



**Figura 3.27. Gráfico de linhas que apresenta a variação da glicose média em função da idade.**

## 3.7. Tendências em Ciência de Dados na Saúde

As ferramentas e técnicas exploradas ao longo deste material, como limpeza de dados, análise exploratória e visualização com Seaborn e Plotly, representam a base prática da ciência de dados aplicada à saúde. Essas habilidades estão no centro de diversas tendências emergentes que vêm transformando o setor. A seguir, apresentamos as principais tendências em ciência de dados aplicada à saúde.

O uso de Inteligência Artificial e Aprendizado de Máquina na saúde tem ganhado destaque por seu potencial em transformar o cuidado com o paciente [Provost and Fawcett 2013]. Algoritmos são capazes de analisar grandes volumes de dados clínicos, laboratoriais e de imagem para identificar padrões que escapam ao olhar humano. Técnicas de aprendizado supervisionado, não supervisionado e aprendizado profundo (deep learning) têm sido amplamente utilizadas em tarefas como detecção precoce de doenças, classificação de imagens

médicas (por exemplo, radiografias e ressonâncias), predição de readmissões hospitalares e estratificação de risco [Moura et al. 2023, Teles et al. 2025]. A capacidade preditiva dessas ferramentas contribui para diagnósticos mais rápidos, precisos e personalizados.

A popularização de dispositivos vestíveis (wearables), como smartwatches e sensores biomédicos, tem permitido a coleta contínua de dados fisiológicos em tempo real [Orphanidou 2019, Moura et al. 2023]. Esses dispositivos capturam informações como batimentos cardíacos, nível de atividade física, qualidade do sono e níveis de glicose. Integrados a plataformas digitais de monitoramento, esses dados podem ser analisados para identificar mudanças no estado de saúde, prevenindo complicações clínicas e permitindo intervenções precoces [Moura et al. 2022]. A saúde digital amplia o alcance do cuidado e promove um modelo mais proativo e centrado no paciente.

A ciência de dados na saúde enfrenta o desafio de lidar com dados oriundos de múltiplas fontes: prontuários eletrônicos, exames laboratoriais, sensores de dispositivos, bases genômicas e dados populacionais [Awrahman et al. 2022]. Esses dados são, muitas vezes, heterogêneos em formato, granularidade e qualidade. A integração dessas fontes permite uma visão mais completa e abrangente do paciente e da população, essencial para análises preditivas, avaliação de intervenções e formulação de estratégias de saúde. Ferramentas de big data e pipelines de processamento são fundamentais nesse contexto.

Modelos preditivos que integram dados clínicos, genômicos e ambientais têm sido amplamente empregados para antecipar a ocorrência de doenças, prever desfechos terapêuticos e estimar riscos individuais à saúde. [Shailaja et al. 2018]. Ao considerar variáveis específicas de cada paciente, é possível propor tratamentos personalizados que maximizem a eficácia e reduzam efeitos adversos. Essa abordagem, conhecida como medicina personalizada ou de precisão, é uma das mais promissoras na interseção entre ciência de dados e saúde [Collins and Varmus 2015]. A modelagem estatística, regressões e algoritmos de aprendizado de máquina são ferramentas centrais nesse processo.

Com o aumento do uso de dados pessoais na saúde, surgem questões críticas relacionadas à ética e à privacidade. O compartilhamento e a análise de informações sensíveis exigem o cumprimento de normativas como a LGPD no Brasil e o GDPR na Europa [Taddeo and Floridi 2018]. É fundamental garantir a anonimização dos dados, o consentimento informado dos pacientes e a transparência nos processos algorítmicos. Além disso, há preocupações sobre vieses em modelos de IA e o impacto ético de decisões automatizadas na saúde. A regulação eficaz é essencial para assegurar o uso responsável e justo da ciência de dados.

## 3.8. Conclusões

Este capítulo de livro forneceu uma introdução abrangente à aplicação de técnicas de ciência de dados na área da saúde, abordando desde a manipulação básica de dados até a criação de gráficos interativos para visualização de padrões e insights. Começamos com a fundamentação do Python, essencial para a construção de um raciocínio lógico sólido e para a criação de soluções eficazes na análise de dados. A compreensão de variáveis, operadores e estruturas condicionais proporcionou uma base essencial para o processamento de dados em saúde.

A manipulação de dados com o Pandas foi explorada de forma prática, permitindo ao leitor entender como selecionar, agregar e filtrar informações relevantes. A análise de dados de saúde, como no estudo de pacientes com risco de diabetes, foi facilitada pela utilização de dataframes, onde cada variável poderia ser manipulada de maneira eficiente e concisa. A introdução a funções de agregação, agrupamento e ordenação, por exemplo, proporcionou uma visão detalhada das técnicas necessárias para explorar grandes volumes de dados e extrair informações úteis.

A seção dedicada à limpeza e tratamento de dados destacou a importância de garantir a qualidade dos dados antes de prosseguir para a análise. O uso de técnicas como a remoção de duplicatas, o tratamento de valores ausentes e a gestão de outliers assegurou que os resultados finais fossem baseados em dados confiáveis e representativos.

Na parte de visualização de dados, a escolha de gráficos adequados, utilizando bibliotecas como Seaborn e Plotly, permitiu a criação de representações claras e informativas. Os gráficos de dispersão, histograma e mapa de calor mostraram como explorar a relação entre variáveis, enquanto os gráficos interativos de Plotly, como o gráfico de pizza e o gráfico Sunburst, possibilitaram uma exploração mais dinâmica dos dados. Esses recursos são essenciais para a comunicação de resultados, especialmente em estudos clínicos, onde a compreensão rápida e precisa dos dados pode ter implicações significativas para a saúde pública.

Ao longo do capítulo, o uso de datasets reais e simulados proporcionou uma compreensão prática das técnicas aplicadas à análise de dados em saúde. O dataset sobre diabetes, por exemplo, ilustrou como diferentes fatores podem impactar o risco de desenvolver doenças, enquanto o estudo de pacientes com histórico de AVC evidenciou a importância de se ter uma visão holística das condições de saúde ao realizar análises.

Por fim, discutimos sobre as tendências em ciência de dados na saúde, destacando como esta área tem impulsionado avanços importantes na saúde, por meio de tecnologias como inteligência artificial, dispositivos vestíveis e modelos preditivos personalizados. Esses recursos ampliam a precisão e a eficiência do cuidado, mas também exigem atenção a questões éticas e regulatórias, reforçando a importância de uma atuação crítica e responsável na área.

Em suma, este capítulo serve como um guia introdutório valioso para aqueles interessados em aplicar técnicas de ciência de dados no campo da saúde. Ao combinar teoria, ferramentas práticas e exemplos do mundo real, os leitores são equipados com as habilidades necessárias para começar a explorar, analisar e comunicar dados de saúde de maneira eficaz e eficiente.

## Referências

[Arellano et al. 2018] Arellano, A. M., Dai, W., Wang, S., Jiang, X., and Ohno-Machado, L. (2018). Privacy policy and technology in biomedical data science. *Annual review of biomedical data science*, 1(1):115–129.

[Awrahman et al. 2022] Awrahman, B. J., Aziz Fatah, C., and Hamaamin, M. Y. (2022). A review of the role and challenges of big data in healthcare informatics and analytics. *Computational intelligence and neuroscience*, 2022(1):5317760.

[Bao et al. 2019] Bao, Y., Chen, Z., Wei, S., Xu, Y., Tang, Z., and Li, H. (2019). The state of the art of data science and engineering in structural health monitoring. *Engineering*, 5(2):234–242.

[Chattu 2021] Chattu, V. K. (2021). A review of artificial intelligence, big data, and blockchain technology applications in medicine and global health. *Big Data and Cognitive Computing*, 5(3):41.

[Collins and Varmus 2015] Collins, F. S. and Varmus, H. (2015). A new initiative on precision medicine. *New England journal of medicine*, 372(9):793–795.

[Latif et al. 2020] Latif, S., Usman, M., Manzoor, S., Iqbal, W., Qadir, J., Tyson, G., Castro, I., Razi, A., Boulos, M. N. K., Weller, A., et al. (2020). Leveraging data science to combat covid-19: A comprehensive review. *IEEE Transactions on Artificial Intelligence*, 1(1):85–103.

[Moura et al. 2022] Moura, I., Teles, A., Coutinho, L., and Silva, F. (2022). Towards identifying context-enriched multimodal behavioral patterns for digital phenotyping of human behaviors. *Future Generation Computer Systems*, 131:227–239.

[Moura et al. 2023] Moura, I., Teles, A., Viana, D., Marques, J., Coutinho, L., and Silva, F. (2023). Digital phenotyping of mental health using multimodal sensing of multiple situations of interest: A systematic literature review. *Journal of Biomedical Informatics*, 138:104278.

[O'connor 2018] O'connor, S. (2018). Big data and data science in health care: What nurses and midwives need to know. *Journal of Clinical Nursing*.

[Orphanidou 2019] Orphanidou, C. (2019). A review of big data applications of physiological signal data. *Biophysical reviews*, 11(1):83–87.

[Provost and Fawcett 2013] Provost, F. and Fawcett, T. (2013). *Data Science for Business: What you need to know about data mining and data-analytic thinking*. "O'Reilly Media, Inc.".

[Shailaja et al. 2018] Shailaja, K., Seetharamulu, B., and Jabbar, M. (2018). Machine learning in healthcare: A review. In *2018 Second international conference on electronics, communication and aerospace technology (ICECA)*, pages 910–914. IEEE.

[Subrahmanya et al. 2022] Subrahmanya, S. V. G., Shetty, D. K., Patil, V., Hameed, B. Z., Paul, R., Smriti, K., Naik, N., and Somani, B. K. (2022). The role of data science in healthcare advancements: applications, benefits, and future prospects. *Irish Journal of Medical Science (1971-)*, 191(4):1473–1483.

[Taddeo and Floridi 2018] Taddeo, M. and Floridi, L. (2018). How ai can be a force for good. *Science*, 361(6404):751–752.

[Talwar et al. 2023] Talwar, R., Sharma, M., Singh, H., and Sagar, P. (2023). A comparative analysis on image processing-based algorithms and approaches in healthcare. In *Handbook of Research on Thrust Technologies' Effect on Image Processing*, pages 1–14. IGI Global.

[Teles et al. 2025] Teles, A. S., de Moura, I. R., Silva, F., Roberts, A., and Stahl, D. (2025). Ehr-based prediction modelling meets multimodal deep learning: A systematic review of structured and textual data fusion methods. *Information Fusion*, page 102981.