Exploração e Análise de Dados usando Python

Material Didático do Projeto +CiênciaSJC

Jonas Lucas Durão

Julho de 2025

Conteúdo

1 Introdução à Ciência de Dados

Este material didático foi desenvolvido para o projeto **+CiênciaSJC**, com o tema "Exploração e Análise de Dados usando Python". O objetivo é introduzir conceitos fundamentais de ciência de dados, modelagem computacional e aprendizado de máquina, utilizando a linguagem Python e a biblioteca Pandas desde o início do processo de aprendizagem.

Definição

A **Ciência de Dados** é um campo multidisciplinar que envolve métodos científicos, processos e sistemas para extrair conhecimento ou *insights* de dados em diversas formas, sejam eles estruturados ou não. Ela combina áreas como estatística, matemática, ciência da computação e conhecimento de domínio para analisar dados e tomar decisões informadas.

2 Módulo 1: Estatística e Visualização de Dados

O Módulo 1 tem como objetivo explorar conceitos estatísticos e ensinar a visualizar dados de forma clara e intuitiva. As aulas abordam a estatística descritiva e a introdução à visualização de dados.

2.1 Aula 01: Revelando Padrões com Dados – Análise Real com Python

2.1.1 Objetivo da Aula

Ao final desta aula, o aluno será capaz de:

- Mostrar como dados reais podem revelar padrões e informações valiosas.
- Ensinar as principais métricas estatísticas (média, mediana, moda, variância) de forma prática.
- Introduzir a biblioteca Pandas de uma maneira intuitiva.
- Despertar a curiosidade para o poder da análise de dados.

2.1.2 O que são Dados?

Definição

Dados são o combustível da Ciência de Dados. Eles podem ser definidos como fatos ou informações, processados ou armazenados, que representam uma característica ou valor.

Tipos de Dados Os dados podem ser categorizados de diversas formas, sendo as principais:

- **Dados Qualitativos (Categóricos):** Representam características ou qualidades que não podem ser medidas numericamente, mas podem ser classificadas em categorias.
  - *Nominais:* Não possuem ordem natural
    Exemplos: cores, categorias de despesa como "Alimentação", "Moradia"
  - *Ordinais:* Possuem uma ordem ou hierarquia
    Exemplos: nível de escolaridade – "Ensino Médio", "Graduação"
- **Dados Quantitativos (Numéricos):** Representam quantidades que podem ser medidas.
  - *Discretos:* Resultam de contagens e assumem valores inteiros
    Exemplos: número de filhos, quantidade de itens comprados
  - *Contínuos:* Resultam de medições e podem assumir qualquer valor dentro de um intervalo
    Exemplos: altura, peso, valores de despesas como "R$45,50"

Importante

A distinção entre esses tipos é crucial, pois ela determina quais métodos estatísticos e tipos de visualização são apropriados.

2.1.3 Estatística Descritiva: Primeiras Medidas

Definição

A **estatística descritiva** é o ramo da estatística que tem como objetivo descrever, resumir e organizar os dados de forma clara e concisa, utilizando medidas de tendência central e de dispersão.

Medidas de Tendência Central São medidas que indicam o ponto central ou típico de um conjunto de dados.

- **Média ($\bar{x}$):** A média aritmética é a soma de todos os valores de um conjunto de dados dividida pelo número de elementos.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{1}$$

Importante

A média é sensível a valores extremos (*outliers*), o que pode distorcer a percepção da centralidade dos dados.

- **Mediana:** É o valor central de um conjunto de dados quando os dados são ordenados em ordem crescente ou decrescente. Se o número de observações (n) for ímpar, a mediana é o valor do meio. Se n for par, a mediana é a média dos dois valores centrais.

Importante

A mediana é robusta a *outliers*.

- **Moda:** É o valor que aparece com maior frequência em um conjunto de dados. Um conjunto de dados pode ter uma moda (unimodal), várias modas (multimodal) ou nenhuma moda.

Medidas de Dispersão São medidas que indicam o grau de variabilidade ou espalhamento dos dados em torno de uma medida de tendência central.

- **Variância (σ^2):** Mede a dispersão dos dados em relação à média. Um valor de variância alto indica que os dados estão espalhados, enquanto um valor baixo indica que estão agrupados em torno da média.

Para uma população, a fórmula é:

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N} \tag{2}$$

Para uma amostra, utiliza-se $n-1$ no denominador para corrigir um viés:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1} \tag{3}$$

Onde N é o tamanho da população e μ é a média da população.

- **Desvio Padrão (σ):** É a raiz quadrada da variância. Ele indica, em média, o quão distantes os valores estão da média e é mais fácil de interpretar do que a variância, pois está na mesma unidade de medida dos dados.

$$\sigma = \sqrt{\sigma^2} \tag{4}$$

- **Coeficiente de Variação (CV):** É uma medida de dispersão relativa, expressa como uma porcentagem. É a razão entre o desvio padrão e a média. É útil para comparar a variabilidade de diferentes conjuntos de dados, mesmo que tenham médias muito diferentes.

$$CV = \frac{\sigma}{\bar{x}} \times 100\% \tag{5}$$

2.1.4 Entendendo as Estruturas de Dados com Pandas

Definição

Pandas é a biblioteca mais utilizada em Python para manipulação e análise de dados. Ela oferece estruturas de dados otimizadas para trabalhar com dados tabulares.

`DataFrame` Um `DataFrame` é uma estrutura de dados bidimensional, semelhante a uma tabela de banco de dados ou uma planilha. Ele possui linhas e colunas rotuladas, e cada coluna pode ter um tipo de dado diferente. É a principal estrutura para armazenar e organizar dados em Pandas.

`Series` Uma `Series` é um objeto unidimensional, como uma lista, array ou vetor, que pode conter qualquer tipo de dado (inteiros, strings, floats, objetos Python, etc.). Cada elemento em uma `Series` tem um rótulo de índice, que pode ser numérico ou textual. Uma coluna de um `DataFrame` é, na verdade, uma `Series`.

Função	Descrição
`df.head()`	Exibe as primeiras n linhas do DataFrame (padrão: 5)
`df.info()`	Resumo do DataFrame: tipos de dados, valores nulos
`df.describe()`	Estatísticas descritivas das colunas numéricas

Tabela 1: Principais funções do Pandas para exploração inicial

Principais Funções para Exploração Inicial

- `df.head()`: Exibe as primeiras n linhas do `DataFrame` (por padrão, 5). Útil para uma inspeção rápida da estrutura e conteúdo dos dados.
- `df.info()`: Fornece um resumo conciso do `DataFrame`, incluindo o número de entradas, o número de colunas, os tipos de dados de cada coluna (Dtype) e a contagem de valores não nulos. Essencial para verificar a completude e os tipos de dados.
- `df.describe()`: Gera estatísticas descritivas para as colunas numéricas do `DataFrame`, como contagem (`count`), média (`mean`), desvio padrão (`std`), mínimo (`min`), quartis (`25%`, `50%` – mediana, `75%`) e máximo (`max`). Ajuda a obter um panorama rápido da distribuição dos dados.

2.1.5 Pré-processamento e Limpeza de Dados

Importante

Dados do mundo real raramente são "limpos" e prontos para análise. O pré-processamento e a limpeza de dados são etapas cruciais para garantir a qualidade e a usabilidade dos dados.

Tratamento de Tipos de Dados É comum que dados numéricos sejam importados como texto (strings), ou datas como objetos genéricos. A conversão para o tipo correto é fundamental para realizar cálculos e análises adequadas:

- `pd.to_datetime()`: Converte strings para o formato de data e hora
- `pd.to_numeric()`: Converte strings ou outros tipos para números

Exemplo

Para converter valores monetários como "R$45,50" para formato numérico:

```
# Remover o simbolo R$
df['Valor (R$)'] = df['Valor (R$)'].astype(str).str.replace('R$', '', regex=False)

# Remover pontos (separador de milhares)
df['Valor (R$)'] = df['Valor (R$)'].str.replace('.', '', regex=False)

# Trocar virgula por ponto (separador decimal)
df['Valor (R$)'] = df['Valor (R$)'].str.replace(',', '.')

# Converter para numerico
df['Valor (R$)'] = pd.to_numeric(df['Valor (R$)'])
```

Listing 1: Conversão de valores monetarios

Tratamento de Valores Ausentes (*Missing Values*) Valores ausentes (frequentemente representados como "NaN" – Not a Number) podem distorcer análises. As estratégias comuns incluem:

- `df.dropna()`: Remove linhas ou colunas que contêm valores ausentes
- `df.fillna()`: Preenche valores ausentes com um valor específico (ex: 0, a média, a mediana, ou o valor anterior/posterior)

Importante

Embora as aulas abordadas não detalhem estas funções, elas serão abordadas com mais detalhes no próximo módulo.

2.1.6 Insights da Análise de Dados

A análise de dados não se resume a calcular números, mas a extrair informações valiosas e padrões que podem guiar decisões. Através da estatística descritiva, podemos obter *insights* como:

- **Despesas Fixas Dominantes:** Identificação de gastos recorrentes e de alto valor (como aluguel, plano de saúde) que consomem grande parte do orçamento. A média

pode ser puxada para cima por esses valores.

- **Relevância de Gastos Frequentes (Pequenas Compras):** Os pequenos gastos diários (café, passagem de ônibus) parecem inofensivos, mas sua frequência pode levar a um montante considerável. A **moda** e a **mediana** são mais indicativas do valor típico de um gasto frequente, em contraste com a média que pode ser influenciada por gastos muito altos e pontuais.
- **O Perigo da Média:** Vimos que a média pode ser enganosa na presença de *outliers*. A mediana e o desvio padrão/coeficiente de variação nos dão uma visão muito mais precisa da centralidade e da dispersão dos gastos por categoria.

2.2 Aula 02: Plotagem de Gráficos e Visualização de Dados

2.2.1 Objetivo da Aula

Ao final desta aula, o aluno será capaz de:

- Aprofundar nos conceitos de visualização de dados usando Python.
- Aprender a criar diferentes tipos de gráficos (histogramas, box plots e gráficos de pizza) para extrair insights.
- Entender como agrupar e consolidar dados para uma análise mais clara.
- Desenvolver a capacidade de interpretar gráficos para tomar decisões financeiras informadas.

2.2.2 A Importância da Visualização de Dados

Definição

A **visualização de dados** é a arte e a ciência de representar informações e dados de forma gráfica.

Em um mundo com volume crescente de dados, a visualização se torna uma ferramenta indispensável porque:

- **Facilita a Compreensão:** Permite que padrões, tendências e *outliers* sejam identificados muito mais rapidamente do que em tabelas de números
- **Suporta a Tomada de Decisão:** Gráficos claros e informativos podem revelar *insights* que guiam estratégias e ações

- **Comunicação Efetiva:** Ajuda a comunicar descobertas complexas a públicos não técnicos

Em Python, as bibliotecas `Matplotlib` e `Seaborn` são as ferramentas padrão para criar visualizações de alta qualidade.

- `Matplotlib`**:** É uma biblioteca de plotagem 2D para Python, considerada a base para muitas outras ferramentas de visualização. Oferece controle granular sobre os elementos do gráfico.
- `Seaborn`**:** Construída sobre o `Matplotlib`, o `Seaborn` oferece uma interface de alto nível para criar gráficos estatísticos atraentes e informativos. É especialmente útil para explorar relações entre múltiplas variáveis e possui belos temas padrão.

2.2.3 Tipos de Gráficos Essenciais e Suas Aplicações

Histograma

Definição

O **histograma** é uma representação gráfica da distribuição de frequência de um conjunto de dados numéricos. Ele divide os dados em intervalos (*bins*) e mostra quantos pontos de dados caem em cada intervalo através da altura das barras.

É excelente para entender a forma, a centralidade e a dispersão de uma única variável.

KDE (Kernel Density Estimate): Frequentemente, um histograma é acompanhado por uma linha sobre as barras, que é a Estimativa de Densidade de Kernel (KDE). A KDE é uma função de suavização que cria uma representação contínua da distribuição de dados. Ela ajuda a visualizar a forma subjacente da distribuição de forma mais suave, identificando picos e vales sem as discretizações das barras do histograma.

Box Plot (Diagrama de Caixa)

Definição

O **Box Plot** é uma ferramenta estatística poderosa para visualizar a distribuição de um conjunto de dados numéricos, identificar sua simetria, dispersão e a presença de *outliers* (valores atípicos).

Ele exibe cinco medidas resumidas, conhecidas como “cinco números”:

1. **Mediana** (linha central da caixa): O valor que divide o conjunto de dados em duas metades iguais (percentil 50).
2. **Primeiro Quartil (Q1)** (base da caixa): Corresponde ao percentil 25, abaixo do qual estão 25% dos dados.
3. **Terceiro Quartil (Q3)** (topo da caixa): Corresponde ao percentil 75, abaixo do qual estão 75% dos dados.

- A altura da caixa representa o **Intervalo Interquartil (IQR = Q3 - Q1)**, que contém os 50% centrais dos dados
- Os "bigodes" (*whiskers*) se estendem do Q1 e Q3 até os valores mínimo e máximo dentro de $1.5 \times IQR$
- Pontos individuais fora dos bigodes são considerados "outliers", indicando valores que se desviam significativamente da maioria dos dados

Gráfico de Pizza (Pie Chart)

Definição

O **gráfico de pizza** é utilizado para mostrar a proporção de cada categoria em relação a um todo. Cada "fatia" do círculo representa a porcentagem ou a frequência relativa de uma categoria sobre o total.

É mais eficaz quando o número de categorias é pequeno (geralmente não mais que 5-7), pois muitas fatias pequenas podem dificultar a leitura e a comparação visual.

Agrupamento de Categorias em "Outros": Para melhorar a legibilidade e evitar a aglomeração de rótulos em gráficos de pizza com muitas categorias, é uma prática comum agrupar todas as categorias que representam uma porcentagem inferior a um determinado limite (ex: 4% ou 7% do total) em uma única fatia rotulada como "Outros". Isso simplifica a visualização e direciona a atenção para as categorias mais relevantes.

Gráfico de Barras (Bar Chart)

Definição

O **gráfico de barras** é ideal para comparar os valores absolutos ou totais entre diferentes categorias. As barras podem ser horizontais ou verticais, e seu comprimento ou altura é proporcional ao valor que representam.

É uma ótima ferramenta para visualizar rapidamente quais categorias têm os maiores ou menores valores.

2.2.4 Interpretação e Tomada de Decisão com Visualizações

A visualização de dados não é apenas sobre criar belos gráficos; é sobre extrair significado e guiar a ação. Por meio dos gráficos, é possível:

- **Identificar Maiores e Menores Gastos:** Gráficos de barras e pizza revelam rapidamente onde o dinheiro está sendo mais ou menos gasto, permitindo a priorização de áreas para economia
- **Compreender a Distribuição de Valores:** O histograma e o box plot oferecem uma visão sobre a frequência de gastos, a existência de picos e se os gastos são consistentes ou variados
- **Detectar Anomalias e Outliers:** O box plot é particularmente útil para identificar gastos atípicos que merecem investigação, como uma compra inesperadamente alta
- **Basear Decisões em Dados:** Com base nessas informações visuais, é possível tomar decisões mais informadas sobre como otimizar o orçamento, cortar gastos desnecessários ou alocar recursos de forma mais eficiente

3 Módulo 2: Limpeza e Preparação de Dados

O Módulo 2 tem como objetivo ensinar técnicas fundamentais para preparar dados para análise e modelagem.

3.1 Aula 1: Tratamento e Manipulação de Dados

3.1.1 Objetivo da Aula

- Ensinar técnicas fundamentais para preparar dados para análise e modelagem.
- Aprender a tratar valores ausentes e corrigir tipos de dados.
- Dominar a manipulação de DataFrames: filtragem, ordenação e criação de novas colunas.
- Realizar um projeto prático de preparação de um dataset sobre empregos e salários.

3.1.2 A Importância da Limpeza e Preparação de Dados

Importante

A limpeza e preparação de dados são etapas cruciais no processo de Ciência de Dados. Dados brutos raramente estão em um formato ideal para análise, podendo conter erros, inconsistências, valores ausentes ou formatos inadequados. Um bom pré-processamento garante que os modelos subsequentes sejam construídos sobre uma base de dados confiável e de alta qualidade.

Valores Ausentes (*Missing Values*) Valores ausentes são dados que não foram registrados ou estão indisponíveis. Eles podem ocorrer por diversos motivos (erros de entrada, falhas de coleta, dados não aplicáveis) e, se não forem tratados adequadamente, podem levar a análises incorretas ou falhas em modelos.

- **Identificação:** Antes de tratar, é preciso identificar onde estão os valores ausentes. Funções como `df.isnull().sum()` são úteis para verificar a contagem de valores nulos por coluna.
- **Estratégias de Tratamento:**
  - `df.dropna()`: Remove linhas ou colunas que contêm valores ausentes.

    Importante

    Use com cautela, pois pode resultar em perda significativa de dados se houver muitos valores ausentes.
  - `df.fillna()`: Preenche valores ausentes com um valor específico.

Exemplo

Preencher com um valor constante (ex: 0, 'Desconhecido'):

```
df['Coluna'].fillna(0, inplace=True)
```

Listing 2: Preencher valores ausentes com um valor constante

Preencher com a média ou mediana da coluna (para dados numéricos):

```
df['Coluna_Numerica'].fillna(df['Coluna_Numerica'].
    mean(), inplace=True)
df['Coluna_Numerica'].fillna(df['Coluna_Numerica'].
    median(), inplace=True)
```

Listing 3: Preencher valores ausentes com a média ou mediana

Preencher com o valor anterior ou posterior (útil para séries temporais):

```
df['Coluna'].fillna(method='ffill', inplace=True) %
    Forward fill
df['Coluna'].fillna(method='bfill', inplace=True) %
    Backward fill
```

Listing 4: Preencher valores ausentes com o valor anterior ou posterior

Correção de Tipos de Dados Garantir que cada coluna tenha o tipo de dado correto (numérico, texto, data, booleano) é fundamental para realizar operações e análises adequadas.

- **Verificação:** `df.info()` ou `df.dtypes` mostram os tipos de dados atuais.
- **Conversão:**
  - `df['Coluna'].astype(tipo)`: Converte uma coluna para um tipo específico (ex: `int`, `float`, `str`).
  - `pd.to_datetime()`: Para converter colunas de data.
  - `pd.to_numeric()`: Para converter colunas para tipos numéricos, tratando erros.

Exemplo

```
df['Coluna_Texto'].astype(str)
df['Coluna_Booleana'].astype(bool)
```

Listing 5: Conversão de tipos de dados

Normalização e Padronização (Transformação de Escala) Em alguns casos, especialmente para modelos de Machine Learning, é importante que as colunas numéricas estejam em uma escala semelhante.

- **Normalização (Min-Max Scaling):** Escala os dados para um intervalo fixo (geralmente entre 0 e 1).

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- **Padronização (Z-score Standardization):** Transforma os dados para ter média 0 e desvio padrão 1.

$$X_{std} = \frac{X - \mu}{\sigma}$$

Importante

Embora importantes, essas técnicas serão aprofundadas no Módulo 4 (Machine Learning).

3.1.3 Manipulação de DataFrames

Pandas oferece poderosas ferramentas para manipular DataFrames, permitindo reorganizar, filtrar e criar novas informações.

Filtragem de Dados Selecionar subconjuntos de dados com base em condições específicas.

Exemplo

```
# Selecionar linhas onde a Categoria    'Alimenta  o'
df_alimentacao = df[df['Categoria'] == 'Alimenta  o']

# Selecionar linhas com Valor (R$) maior que 100
df_gastos_altos = df[df['Valor (R$)'] > 100]

# Combinar condi  es
df_condicoes = df[(df['Categoria'] == 'Transporte') & (df['
    Valor (R$)'] < 50)]
```

Listing 6: Filtragem de dados

Ordenação de Dados Organizar o DataFrame com base nos valores de uma ou mais colunas.

Exemplo

```
# Ordenar por Valor (R$) em ordem crescente
df_ordenado_valor = df.sort_values(by='Valor (R$)')

# Ordenar por Data em ordem decrescente
df_ordenado_data = df.sort_values(by='Data', ascending=False)
```

Listing 7: Ordenação de dados

Criação de Novas Colunas Derivar novas informações a partir de colunas existentes.

Exemplo

```
# Criar uma coluna 'Mes' a partir da coluna 'Data'
df['Mes'] = df['Data'].dt.month

# Criar uma coluna 'Custo_Por_Item' (se houver 'Quantidade' e 'Valor')
# df['Custo_Por_Item'] = df['Valor'] / df['Quantidade']

# Criar uma coluna bin ria 'Gasto_Alto'
df['Gasto_Alto'] = df['Valor (R$)'] > df['Valor (R$)'].median()
```

Listing 8: Criação de novas colunas

Agregação de Dados Resumir dados usando funções estatísticas (soma, média, contagem) agrupando por uma ou mais categorias.

Exemplo

```
# Gasto total por categoria
gastos_por_categoria = df.groupby('Categoria')['Valor (R$)'].sum()

# Consumo m dio de energia por m s
# consumo_medio_mes = df_energia.groupby('Mes')['consumo_kwh'].mean()
```

Listing 9: Agregação de dados

4 Módulo 3: Introdução à Modelagem Computacional e Simulação

4.1 Aula 1: Conceitos de Modelagem e Crescimento Populacional

4.1.1 Objetivo da Aula

- Apresentar os conceitos básicos de modelagem computacional.

- Explorar modelos simples para simular fenômenos da vida real, como o crescimento populacional.
- Entender a importância de testar diferentes cenários em modelos.
- Discutir as limitações dos modelos e a necessidade de validação.

4.1.2 O que é Modelagem Computacional?

Definição

A **modelagem computacional** é a arte de criar representações simplificadas de sistemas ou fenômenos do mundo real usando matemática, lógica e programação.

Essas representações, ou "modelos", nos permitem:

- **Simular:** Recriar o comportamento de um sistema ao longo do tempo, sem a necessidade de esperar que o fenômeno real ocorra. Isso é útil para entender como diferentes variáveis interagem.
- **Prever:** Estimar comportamentos futuros ou resultados com base em dados históricos e suposições sobre o futuro.
- **Testar Cenários:** Avaliar o impacto de diferentes condições ou decisões sem riscos, permitindo a experimentação em um ambiente virtual.
- **Entender Padrões Complexos:** Simplificar a complexidade do mundo real para identificar e analisar padrões que seriam difíceis de perceber de outra forma.

Exemplos de Aplicações:

- Previsão do tempo.
- Simulação da propagação de doenças (como pandemias).
- Projeções de crescimento populacional.
- Simulações financeiras para investimentos.
- Design de produtos e engenharia (testando protótipos virtualmente).

4.1.3 Modelos de Crescimento Populacional

Vamos explorar dois modelos fundamentais para entender como as populações podem crescer ao longo do tempo: o crescimento linear e o crescimento exponencial.

Crescimento Linear Simples Neste modelo, assume-se que a população cresce por uma quantidade **constante** a cada período de tempo (por exemplo, a cada ano). A taxa de crescimento é um valor absoluto que é adicionado à população existente. A fórmula para o crescimento linear é:

$$P_t = P_0 + (k \times t)$$

Onde:

- P_t é a população no tempo t.
- P_0 é a população inicial (no tempo $t = 0$).
- k é a taxa de crescimento linear (quantidade de indivíduos adicionados por unidade de tempo).
- t é o tempo (número de períodos).

Importante

Este modelo é uma simplificação e raramente reflete o crescimento populacional real por longos períodos, pois as populações tendem a crescer com base em seu tamanho atual.

Crescimento Exponencial O modelo de crescimento exponencial é mais realista para muitas populações, especialmente em estágios iniciais, onde os recursos não são limitantes. Ele assume que a população cresce a uma taxa **proporcional** ao seu tamanho atual. Ou seja, quanto maior a população, maior o número absoluto de novos indivíduos adicionados. A fórmula básica para o crescimento exponencial é:

$$P_t = P_0 \times (1 + r)^t$$

Onde:

- P_t é a população no tempo t.
- P_0 é a população inicial.
- r é a taxa de crescimento percentual por período (ex: 0.015 para 1.5%).
- t é o tempo (número de períodos).

Em cenários mais complexos, o crescimento exponencial pode incluir um *fator de desaceleração*, que simula a redução da taxa de crescimento ao longo do tempo devido a fatores como limitação de recursos, aumento da competição ou políticas de controle populacional.

Por exemplo, a taxa r pode ser multiplicada por um fator menor que 1 a cada período:

$$r_{novo} = r_{antigo} \times \text{fator\_desaceleração}$$

Isso faz com que o crescimento seja mais lento à medida que a população aumenta.

4.1.4 Comparação de Modelos e Teste de Cenários

A modelagem computacional não se limita a criar um único modelo. É fundamental:

- **Comparar Modelos:** Entender as diferenças entre modelos (linear vs. exponencial) e como cada um se comporta sob diferentes suposições. O crescimento exponencial geralmente prevê um aumento muito mais rápido ao longo do tempo do que o linear.
- **Testar Diferentes Cenários:** Variar os parâmetros de entrada do modelo (ex: taxa de crescimento inicial, fator de desaceleração, número de anos de projeção) para observar como as projeções mudam. Isso permite criar cenários otimistas, conservadores ou de estagnação, ajudando na tomada de decisões e no planejamento.
- **Adicionar Fatores Externos:** Modelos mais avançados podem incorporar variáveis adicionais que influenciam o crescimento populacional, como migração (entrada ou saída de pessoas), políticas públicas (incentivos à natalidade ou controle), desenvolvimento econômico e eventos externos (pandemias, crises econômicas). A inclusão desses fatores torna o modelo mais robusto e realista.

4.1.5 Limitações e Validação de Modelos

Importante

É crucial reconhecer que todo modelo é uma simplificação da realidade. As limitações incluem:

- **Simplificações:** Modelos ignoram muitos detalhes e complexidades do mundo real para serem gerenciáveis.
- **Dependência de Suposições:** As projeções são tão boas quanto as suposições e os dados de entrada. Se as suposições forem falhas, as previsões também serão.
- **Validação:** Para aumentar a confiança em um modelo, é essencial validá-lo. Isso envolve comparar as projeções do modelo com dados reais históricos (se disponíveis) ou com dados coletados após a projeção. A validação ajuda a ajustar o modelo e a entender sua precisão.

4.2 Aula 2: Simulação de Consumo Energético e Probabilidade

4.2.1 Objetivo da Aula

- Simular o consumo energético de uma residência, considerando fatores sazonais e aleatórios.
- Introduzir e aplicar conceitos de probabilidade em modelos computacionais.
- Simular cenários de investimento e analisar o risco e retorno.
- Realizar um projeto prático integrado de simulação de gastos e investimentos.

4.2.2 Simulação de Consumo Energético

O consumo de energia elétrica em uma residência é um fenômeno complexo que varia significativamente ao longo do tempo. Um modelo de simulação pode capturar essas variações, permitindo prever custos e planejar o orçamento.

Fatores de Variação do Consumo

- **Consumo Base:** O consumo mínimo e constante da residência (ex: geladeira, iluminação básica).
- **Estação do Ano (Sazonalidade):** O consumo tende a ser maior em meses de verão (devido ao uso de ar condicionado) e, em algumas regiões, em meses de inverno (devido a aquecedores). Fatores sazonais são aplicados como multiplicadores ao consumo base.
- **Dia da Semana e Hora do Dia:** Embora não detalhado no modelo da aula, o consumo real também varia entre dias úteis e fins de semana, e em picos de uso durante o dia (manhã, noite).
- **Fatores Aleatórios:** Variações imprevistas no comportamento dos moradores, falhas de equipamentos, ou eventos climáticos pontuais podem introduzir aleatoriedade no consumo.
- **Tendências Anuais:** O consumo pode ter uma tendência de crescimento ou decréscimo ao longo dos anos (ex: devido à aquisição de novos aparelhos ou à implementação de hábitos mais eficientes).

Um modelo de simulação de consumo energético combina esses fatores para gerar valores realistas de consumo e custo ao longo de um período.

4.2.3 Introdução à Probabilidade em Simulações

Importante

A **probabilidade** nos ajuda a modelar **incertezas** e **variações aleatórias**.

Conceitos Importantes

- **Aleatoriedade:** A capacidade de modelar eventos imprevisíveis, mas que seguem certas distribuições de probabilidade.
- **Distribuições Estatísticas:** Funções matemáticas que descrevem a probabilidade de diferentes resultados ocorrerem.
  - *Distribuição Normal (Gaussiana):* Caracterizada por uma curva em forma de sino, onde a maioria dos valores se concentra em torno da média. É frequentemente usada para modelar variações naturais, como retornos de investimentos ou consumo energético.
  - *Distribuição Uniforme:* Todos os valores dentro de um determinado intervalo têm a mesma probabilidade de ocorrer. Útil para simular eventos onde não há preferência por um valor específico.
  - *Distribuição Exponencial:* Descreve o tempo entre eventos em um processo Poisson, onde os eventos ocorrem continuamente e independentemente a uma taxa média constante.
  - *Distribuição Binomial:* Usada para modelar o número de sucessos em uma sequência de n tentativas independentes, onde cada tentativa tem apenas dois resultados possíveis (sucesso/falha).
- **Simulação de Monte Carlo:** Uma técnica computacional que usa amostragem aleatória para obter resultados numéricos. É particularmente útil para modelar sistemas complexos com muitas variáveis incertas, permitindo que se estime a probabilidade de diferentes resultados ocorrerem. No contexto da aula, ela é usada para simular a variação aleatória no consumo de energia e nos retornos de ações.

4.2.4 Simulação de Investimentos

A simulação de investimentos permite projetar o valor futuro de um investimento, considerando diferentes taxas de retorno e volatilidade. Isso é crucial para entender o risco associado a cada tipo de investimento.

Tipos de Investimento e Características

- **Poupança:** Geralmente, um investimento de baixo risco com uma taxa de retorno fixa e previsível. É um bom ponto de partida para comparação.
- **CDB (Certificado de Depósito Bancário):** Tende a oferecer uma taxa de retorno fixa, mas geralmente mais alta que a poupança, com risco moderado.
- **Ações:** Investimentos de maior risco, mas com potencial de maior retorno. Seus retornos são voláteis e podem ser modelados usando distribuições de probabilidade (ex: distribuição normal para o retorno mensal).

Análise de Risco vs. Retorno

- **Retorno Médio:** O ganho médio esperado de um investimento.
- **Volatilidade (Desvio Padrão):** Mede a dispersão dos retornos de um investimento. Quanto maior a volatilidade, maior o risco.
- **Múltiplas Simulações:** Ao invés de uma única projeção, realizar milhares de simulações (Monte Carlo) para um investimento volátil (como ações) permite visualizar a **distribuição** dos possíveis retornos finais. Isso ajuda a entender o pior e o melhor cenário, bem como a probabilidade de atingir um determinado objetivo financeiro.

4.2.5 Projeto Prático Integrado: Simulação de Gastos Energéticos e Investimentos

Este projeto combina os conceitos de modelagem de consumo energético e simulação de investimentos para criar um cenário mais completo de planejamento financeiro.

Etapas do Projeto

1. **Simulação de Economia de Energia:** Criar diferentes cenários de consumo (normal, com economia, com energia solar) para quantificar a economia potencial em termos de custo mensal.
2. **Cálculo da Economia Média Mensal:** Determinar o valor médio que seria economizado por mês ao adotar uma estratégia de redução de consumo.
3. **Simulação de Investimento da Economia:** Projetar o quanto o dinheiro economizado renderia se fosse investido em diferentes opções (poupança, CDB, ações), aplicando os modelos de simulação de investimentos.

Análise de Sensibilidade e Intervalo de Confiança

- **Análise de Sensibilidade:** Testar como pequenas mudanças nos parâmetros de entrada (ex: percentual de economia de energia, taxas de retorno) afetam os resultados finais do modelo. Isso ajuda a identificar quais variáveis têm o maior impacto e onde focar os esforços de otimização.
- **Intervalo de Confiança:** Para simulações com aleatoriedade (como o retorno de ações), calcular o intervalo de confiança (ex: 95% de IC). Isso fornece um intervalo dentro do qual o valor real do retorno final tem uma alta probabilidade de cair, dando uma medida da incerteza da previsão. Por exemplo, um IC de 95% para o retorno de ações de R$ 76.88 a R$ 5712.84 significa que, em 95% das simulações, o valor final do investimento em ações ficou dentro desse intervalo.

5 Módulo 4: Introdução ao Aprendizado de Máquina (Machine Learning)

5.1 Objetivo do Módulo

Apresentar noções básicas de aprendizado de máquina de forma prática, utilizando ferramentas acessíveis e preparando o terreno para a construção de modelos preditivos.

5.2 Conceitos Iniciais de Machine Learning

Definição

Aprendizado de Máquina (Machine Learning - ML) é um subcampo da inteligência artificial que permite que sistemas aprendam com dados, identifiquem padrões e tomem decisões com o mínimo de intervenção humana. Em vez de ser explicitamente programado para cada tarefa, o algoritmo "aprende"a realizar a tarefa a partir de exemplos.

Tipos de Aprendizado de Máquina

- **Aprendizado Supervisionado:**

Definição

No **aprendizado supervisionado**, o algoritmo é treinado com um conjunto de dados que inclui tanto as "entradas"(características ou *features*) quanto as "saídas"corretas (rótulos ou *labels*). O objetivo é que o modelo aprenda a mapear as entradas para as saídas, de modo que possa prever as saídas para novos dados não vistos.

  - **Regressão:** Usado para prever um valor contínuo (ex: preço de uma casa, gastos mensais, valorização de investimentos).
  - **Classificação:** Usado para prever uma categoria discreta (ex: se um e-mail é spam ou não, tipo de doença, aprovação de crédito).

- **Aprendizado Não Supervisionado:**

Definição

No **aprendizado não supervisionado**, o algoritmo recebe dados sem rótulos predefinidos. O objetivo é encontrar padrões, estruturas ou agrupamentos ocultos nos dados.

  - **Agrupamento (Clustering):** Agrupar pontos de dados semelhantes (ex: segmentação de clientes).
  - **Redução de Dimensionalidade:** Reduzir o número de características, mantendo a maior parte da informação (ex: PCA - Análise de Componentes Principais).

- **Aprendizado por Reforço (Reinforcement Learning):**

Definição

No **aprendizado por reforço**, um agente aprende a tomar decisões em um ambiente para maximizar uma recompensa. O agente não é explicitamente treinado com dados, mas sim por tentativa e erro.

  - **Exemplos:** Carros autônomos, jogos (AlphaGo).

Terminologia Essencial

- **Features (Características):** As variáveis de entrada ou atributos usados para fazer previsões (colunas do DataFrame).

- **Labels (Rótulos):** A variável de saída ou o que estamos tentando prever (a coluna alvo).
- **Modelo:** O algoritmo de Machine Learning treinado que aprendeu os padrões nos dados.
- **Treinamento:** O processo de alimentar o modelo com dados rotulados para que ele aprenda os padrões.
- **Predição/Inferência:** O processo de usar o modelo treinado para fazer previsões em novos dados não vistos.

5.3 Separação e Avaliação de Dados

Importante

Para garantir que um modelo de Machine Learning generalize bem para dados não vistos (ou seja, que ele não apenas memorize os dados de treinamento), é crucial separar o conjunto de dados em subconjuntos de treinamento e teste.

Conjunto de Treinamento e Teste (`train_test_split`)

Definição

O **conjunto de treinamento** é usado para o modelo aprender os padrões. O **conjunto de teste** é usado para avaliar o desempenho do modelo em dados que ele nunca viu antes.

Uma divisão comum é 70-80% para treinamento e 20-30% para teste. A função `train_test_split` do `scikit-learn` é ideal para isso.

Exemplo

```python
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Exemplo de dados (features X e labels y)
X = pd.DataFrame(np.random.rand(100, 5), columns=[f'feature_{i}' for i in range(5)])
y = pd.Series(np.random.randint(0, 2, 100)) # Exemplo de classificação binária

# Separando 80% para treinamento e 20% para teste
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Tamanho do conjunto de treinamento (X): {X_treino.shape}")
print(f"Tamanho do conjunto de teste (X): {X_teste.shape}")
print(f"Tamanho do conjunto de treinamento (y): {y_treino.shape}")
print(f"Tamanho do conjunto de teste (y): {y_teste.shape}")
```

Listing 10: Separação de dados em treinamento e teste

Validação Cruzada (*Cross-validation*)

Definição

A **validação cruzada** é uma técnica mais robusta para avaliar o desempenho do modelo, especialmente em datasets menores. Ela divide os dados em múltiplos subconjuntos (folds), treinando e testando o modelo várias vezes com diferentes combinações.

Importante

Isso ajuda a reduzir o viés da divisão única de treinamento/teste e fornece uma estimativa mais confiável da performance do modelo.

5.4 Modelos Iniciais com scikit-learn

Definição

scikit-learn é uma biblioteca Python de código aberto que implementa uma vasta gama de algoritmos de Machine Learning, além de ferramentas para pré-processamento de dados e avaliação de modelos. É uma das bibliotecas mais populares e acessíveis para ML.

Definição

A **regressão linear** é um modelo supervisionado usado para prever um valor con
tínuo (*label*) com base em uma ou mais variáveis de entrada (*features*). Ela assum
uma relação linear entre as features e o label.

Regressão Linear

A equação de uma regressão linear simples (com uma feature) é:

$$y = \beta_0 + \beta_1 x_1 + \epsilon$$

Onde:

- y é o valor que queremos prever (label).
- β_0 é o intercepto (o valor de y quando x_1 é 0).
- β_1 é o coeficiente da feature x_1 (inclinação da linha).
- x_1 é a feature de entrada.
- ϵ é o termo de erro (o que o modelo não consegue explicar).

O objetivo do treinamento é encontrar os melhores valores para β_0 e β_1 que minimizem o erro entre as previsões do modelo e os valores reais.

Exemplo

```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# Dados de exemplo: rela  o entre horas de estudo e nota
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1) # Horas de estudo
y = np.array([20, 35, 40, 50, 65, 70, 80, 85, 90, 95]) # Notas

# Separar dados
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)

# Criar e treinar o modelo
modelo_rl = LinearRegression()
modelo_rl.fit(X_treino, y_treino)

# Fazer previs es
y_previsao = modelo_rl.predict(X_teste)

print(f"Coeficiente (beta_1): {modelo_rl.coef_[0]:.2f}")
print(f"Intercepto (beta_0): {modelo_rl.intercept_:.2f}")
print(f"Previs es para o conjunto de teste: {y_previsao}")

# Visualizar a regress o (opcional)
plt.scatter(X_treino, y_treino, color='blue', label='Dados de Treino')
plt.scatter(X_teste, y_teste, color='green', label='Dados de Teste')
plt.plot(X, modelo_rl.predict(X), color='red', label='Linha de Regress o')
plt.xlabel("Horas de Estudo")
plt.ylabel("Nota")
plt.title("Regress o Linear: Horas de Estudo vs. Nota")
plt.legend()
plt.show()
```

Listing 11: Exemplo de Regressão Linear com scikit-learn

Definição

Uma **Árvore de Decisão** é um modelo supervisionado que pode ser usado tan
para classificação quanto para regressão. Ela funciona dividindo o conjunto de dad
em subconjuntos menores com base em decisões simples (regras de "se-então") sob
as características. O resultado é uma estrutura em forma de árvore, onde cada
interno representa um teste em uma característica, cada ramo representa o resulta
do teste, e cada nó folha (terminal) representa a decisão final ou o valor previsto

Árvore de Decisão

- **Nós (Nodes):** Pontos de decisão baseados em uma característica.
- **Ramos (Branches):** Os caminhos que levam aos resultados de um teste.
- **Folhas (Leaves):** Os nós finais que contêm a previsão ou classificação.

Importante

Árvores de decisão são fáceis de entender e visualizar, mas podem ser propensas a *overfitting* (memorizar os dados de treinamento) se não forem controladas (ex: limitando a profundidade da árvore).

Exemplo

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Dados de exemplo: caracter sticas do cliente para prever compra (0=N o , 1=Sim)
X = pd.DataFrame({
    'Idade': [25, 30, 45, 20, 50, 35, 60, 28, 40, 22],
    'Renda': [3000, 5000, 7000, 2500, 8000, 6000, 9000, 4000, 6500, 3200],
    'Historico_Compras': [1, 3, 5, 0, 4, 2, 6, 1, 3, 0]
})
y = pd.Series([0, 1, 1, 0, 1, 1, 1, 0, 1, 0]) # Comprou ou N o Comprou

# Separar dados
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)

# Criar e treinar o modelo
modelo_ad = DecisionTreeClassifier(random_state=42)
modelo_ad.fit(X_treino, y_treino)

# Fazer previs es
y_previsao = modelo_ad.predict(X_teste)

print(f"Previs es para o conjunto de teste: {y_previsao}")
print(f"Valores reais do conjunto de teste: {y_teste.values}")
```

Listing 12: Exemplo de Árvore de Decisão para Classificação

Definição

K-Nearest Neighbors (KNN) é um algoritmo de aprendizado não-paramétrico usado para classificação e regressão. Ele é um "ap çoso"porque não constrói um modelo explícito durante a fase de tre sim armazena todo o conjunto de dados de treinamento. Para fazer para um novo ponto de dados, o KNN encontra os "K"pontos de d ximos no conjunto de treinamento (baseado em alguma métrica de d a distância Euclidiana).

KNN (K-Nearest Neighbors)

- **Para Classificação:** O rótulo do novo ponto de dados é determinado pela maioria dos rótulos de seus K vizinhos mais próximos.
- **Para Regressão:** O valor previsto para o novo ponto de dados é a média (ou mediana) dos valores dos rótulos de seus K vizinhos mais próximos.
- **Parâmetro K:** A escolha do valor de K é crucial e afeta o desempenho do modelo. Um K pequeno pode ser sensível a ruídos; um K grande pode suavizar demais as fronteiras de decisão.

Importante

KNN é simples de entender e implementar, mas pode ser computacionalmente caro para grandes datasets e sensível à escala das características.

Exemplo

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Dados de exemplo: caracter sticas de flores para
    classificar a esp cie
X = pd.DataFrame({
    'Comprimento_Sepala': [5.1, 4.9, 6.3, 5.0, 5.5, 6.7, 5.8,
        6.1, 5.3, 5.4],
    'Largura_Sepala': [3.5, 3.0, 3.3, 3.4, 2.3, 3.1, 2.7, 2.8,
        3.7, 3.0]
})
y = pd.Series(['Setosa', 'Setosa', 'Versicolor', 'Setosa', '
    Versicolor', 'Virginica', 'Versicolor', 'Virginica', '
    Setosa', 'Versicolor'])

# Separar dados
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y,
    test_size=0.3, random_state=42)

# Criar e treinar o modelo KNN com K=3
modelo_knn = KNeighborsClassifier(n_neighbors=3)
modelo_knn.fit(X_treino, y_treino)

# Fazer previs es
y_previsao = modelo_knn.predict(X_teste)

print(f"Previs es para o conjunto de teste: {y_previsao}")
print(f"Valores reais do conjunto de teste: {y_teste.values}")
```

Listing 13: Exemplo de KNN para Classificação

5.5 Avaliação de Modelos

Após treinar um modelo, é essencial avaliar seu desempenho para entender o quão bem ele faz previsões e se ele está apto para a tarefa. As métricas de avaliação variam dependendo se o problema é de regressão ou classificação.

Métricas para Regressão Usadas quando o modelo prevê um valor contínuo.

- **Erro Médio Absoluto (MAE - Mean Absolute Error):**

Definição

O **MAE** é a média das diferenças absolutas entre os valores previstos pelo modelo e os valores reais. Ele mede a magnitude média dos erros sem considerar a direção.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Onde y_i é o valor real e $\hat{y}_i$ é o valor previsto. Um MAE menor indica um modelo mais preciso.

- **Erro Quadrático Médio (MSE - Mean Squared Error):**

Definição

O **MSE** é a média dos quadrados das diferenças entre os valores previstos e os reais. Ele penaliza erros maiores mais severamente.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Raiz do Erro Quadrático Médio (RMSE - Root Mean Squared Error):**

Definição

O **RMSE** é a raiz quadrada do MSE. Ele é mais interpretável que o MSE porque está na mesma unidade da variável alvo.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- R^2 **(Coeficiente de Determinação):**

Definição

O R^2 mede a proporção da variância na variável dependente que é previsível a partir das variáveis independentes. Varia de 0 a 1 (ou pode ser negativo para modelos muito ruins).

Importante

Um R^2 mais próximo de 1 indica que o modelo explica uma grande parte da variabilidade dos dados.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Onde $\bar{y}$ é a média dos valores reais.

Exemplo

```python
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np

# Dados de exemplo (horas de estudo vs. nota)
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
y = np.array([20, 35, 40, 50, 65, 70, 80, 85, 90, 95])

X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)

modelo_rl = LinearRegression()
modelo_rl.fit(X_treino, y_treino)
y_previsao = modelo_rl.predict(X_teste)

# Calcular m tricas
mae = mean_absolute_error(y_teste, y_previsao)
r2 = r2_score(y_teste, y_previsao)

print(f"Erro M dio Absoluto (MAE): {mae:.2f}")
print(f"Coeficiente de Determina  o (R^2): {r2:.2f}")
```

Listing 14: Avaliação de Modelo de Regressão

Métricas para Classificação Usadas quando o modelo prevê uma categoria.

- **Matriz de Confusão:**

Definição

A **Matriz de Confusão** é uma tabela que resume o desempenho de um algoritmo de classificação, mostrando o número de previsões corretas e incorretas para cada classe.

Ela é fundamental para entender os tipos de erros que o modelo está cometendo. Para um problema de classificação binária (duas classes: Positivo e Negativo), a matriz tem 4 componentes:

* **Verdadeiro Positivo (VP):** O modelo previu Positivo e era realmente Positivo.
* **Verdadeiro Negativo (VN):** O modelo previu Negativo e era realmente Negativo.
* **Falso Positivo (FP):** O modelo previu Positivo, mas era realmente Negativo (Erro Tipo I).
* **Falso Negativo (FN):** O modelo previu Negativo, mas era realmente Positivo (Erro Tipo II).

	Previsto Positivo	**Previsto Negativo**
Real Positivo	VP	FN
Real Negativo	FP	VN

Tabela 2: Estrutura da Matriz de Confusão

– **Acurácia (Accuracy):**

Definição

A **Acurácia** mede a proporção de previsões corretas em relação ao total de previsões.

$$Acurcia = \frac{VP + VN}{VP + VN + FP + FN}$$

Importante

A acurácia pode ser enganosa em datasets desbalanceados (onde uma classe é muito mais frequente que a outra).

– **Precisão (Precision):**

Definição

A **Precisão** mede a proporção de previsões positivas corretas em relação ao total de previsões positivas feitas pelo modelo. Responde: "Das vezes que previ positivo, quantas estavam corretas?"

$$Preciso = \frac{VP}{VP + FP}$$

– **Revocação (Recall ou Sensibilidade):**

Definição

A **Revocação** mede a proporção de previsões positivas corretas em relação ao total de casos positivos reais. Responde: "Dos casos positivos reais, quantos o modelo conseguiu identificar?"

$$Revocao = \frac{VP}{VP + FN}$$

– **F1-Score:**

Definição

O **F1-Score** é a média harmônica da Precisão e da Revocação. É útil quando há um desequilíbrio entre as classes e você precisa de um equilíbrio entre Precisão e Revocação.

$$F1 - Score = 2 \times \frac{Preciso \times Revocao}{Preciso + Revocao}$$

Exemplo

```python
from sklearn.metrics import confusion_matrix,
    accuracy_score, precision_score, recall_score, f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Dados de exemplo (caracter sticas do cliente para
    prever compra)
X = pd.DataFrame({
    'Idade': [25, 30, 45, 20, 50, 35, 60, 28, 40, 22],
    'Renda': [3000, 5000, 7000, 2500, 8000, 6000, 9000,
        4000, 6500, 3200],
    'Historico_Compras': [1, 3, 5, 0, 4, 2, 6, 1, 3, 0]
})
y = pd.Series([0, 1, 1, 0, 1, 1, 1, 0, 1, 0]) # Comprou ou
    N o Comprou

X_treino, X_teste, y_treino, y_teste = train_test_split(X,
    y, test_size=0.3, random_state=42)

modelo_ad = DecisionTreeClassifier(random_state=42)
modelo_ad.fit(X_treino, y_treino)
y_previsao = modelo_ad.predict(X_teste)

# Calcular m tricas
cm = confusion_matrix(y_teste, y_previsao)
acuracia = accuracy_score(y_teste, y_previsao)
precisao = precision_score(y_teste, y_previsao)
revocacao = recall_score(y_teste, y_previsao)
f1 = f1_score(y_teste, y_previsao)

print(f"Matriz de Confus o:\n{cm}")
print(f"Acur cia: {acuracia:.2f}")
print(f"Precis o: {precisao:.2f}")
print(f"Revoca  o: {revocacao:.2f}")
print(f"F1-Score: {f1:.2f}")
```

Listing 15: Avaliação de Modelo de Classificação

5.6 Projeto Prático: Previsão de Gastos Mensais ou de Valorização de Investimentos

Neste projeto prático, você aplicará os conceitos e modelos aprendidos para resolver um problema real de previsão. Usando os dados de gastos mensais ou informações de investimentos (que podem ser simulados ou reais, como os do Módulo 1 e 3), você construirá um modelo de Machine Learning.

Passos Sugeridos:

1. **Coleta e Preparação de Dados:** Utilize os dados já preparados dos módulos anteriores ou um novo dataset. Certifique-se de que os dados estejam limpos, com tipos corretos e, se necessário, transformados (normalização/padronização).
2. **Definição do Problema:**
   - **Previsão de Gastos Mensais:** O objetivo é prever o valor total de gastos em um determinado mês. Este é um problema de **regressão**. As *features* podem incluir mês, ano, categoria de gasto (se for detalhado), renda, número de pessoas na casa, etc.
   - **Previsão de Valorização de Investimentos:** O objetivo pode ser prever o valor futuro de um investimento (problema de **regressão**) ou classificar se um investimento terá retorno positivo ou negativo (problema de **classificação**). As *features* podem incluir dados históricos do ativo, indicadores econômicos, taxas de juros, etc.
3. **Separação de Dados:** Divida seu dataset em conjuntos de treinamento e teste usando `train_test_split`.
4. **Escolha e Treinamento do Modelo:**
   - Para problemas de **regressão** (previsão de valores contínuos): Experimente `LinearRegression` ou `DecisionTreeRegressor` (do `sklearn.tree`).
   - Para problemas de **classificação** (previsão de categorias): Experimente `DecisionTreeClassifier` ou `KNeighborsClassifier` (do `sklearn.neighbors`).
5. **Avaliação do Modelo:**
   - Para **regressão**: Use `mean_absolute_error` e `r2_score`.
   - Para **classificação**: Use `confusion_matrix`, `accuracy_score`, `precision_score`, `recall_score` e `f1_score`.

6. **Interpretação dos Resultados:** Analise as métricas. O modelo está performando bem? Quais são as principais características que influenciam a previsão?
7. **Otimização (Opcional):** Se o tempo permitir, explore a otimização de hiperparâmetros (ajustar as configurações do modelo) ou a engenharia de características (criar novas features a partir das existentes) para melhorar o desempenho.

6 Conclusão

Este material didático forneceu uma jornada abrangente pela exploração, análise e modelagem computacional de dados. Iniciamos com uma base sólida em **Estatística Descritiva e Visualização de Dados (Módulo 1)**, onde aprendemos a resumir informações e a transformá-las em *insights* acionáveis por meio de gráficos claros e informativos.

Em seguida, mergulhamos na **Modelagem Computacional e Simulação (Módulo 3)**. Na Aula 1, exploramos como criar modelos matemáticos simples para simular fenômenos do mundo real, como o crescimento populacional, e a importância de testar diferentes cenários e entender as limitações dos modelos. Na Aula 2, aprofundamos esses conceitos, aplicando a simulação ao consumo energético e a cenários de investimento. Introduzimos a **probabilidade** como uma ferramenta essencial para lidar com a incerteza e exploramos a **Simulação de Monte Carlo** para analisar o risco e o retorno de investimentos. O projeto prático integrado consolidou esses conhecimentos, mostrando como a modelagem pode ser usada para tomar decisões financeiras mais informadas.

Ao longo desta apostila, você desenvolveu a capacidade de:

- Manipular e pré-processar dados usando a biblioteca Pandas.
- Calcular e interpretar métricas estatísticas essenciais.
- Criar visualizações eficazes para comunicar descobertas.
- Construir e comparar modelos de simulação para diferentes fenômenos.
- Incorporar aleatoriedade e probabilidade em suas análises.
- Avaliar o risco e testar a sensibilidade dos seus modelos.

Com essas habilidades, você está agora apto a abordar problemas complexos do mundo real de uma perspectiva baseada em dados, utilizando o poder da computação para simular, prever e tomar decisões. O próximo passo em sua jornada será a **Introdução ao Aprendizado de Máquina (Módulo 4)**, onde você aprenderá

a construir modelos preditivos que podem identificar padrões ainda mais complexos e automatizar a tomada de decisões. Prepare-se para explorar o fascinante mundo da inteligência artificial!