Pós-Graduação em Inteligência Artificial Generativa e LLMs

PUC-Rio

Sistemas de Apoio à Decisão

Notas de Aula

Vítor Wilher

SAD · Eletiva

Versão 1.0

Resumo. Métodos e sistemas de apoio à decisão aplicados a problemas de negócio.
Palavras-chave: apoio à decisão; modelagem; análise

Notas de estudo elaboradas a partir do material das aulas.

Índice

1 Visão Geral da Disciplina

A disciplina **Sistemas de Apoio à Decisão (SAD)** reúne, em um único percurso, os três pilares técnicos que sustentam a produção de informação para a tomada de decisão nas organizações: a **programação em Python**, a **linguagem SQL** para bancos de dados relacionais e as **ferramentas de visualização e Business Intelligence**, com destaque para o Power BI. A ementa alterna deliberadamente entre esses blocos, de modo que o aluno construa competências em paralelo, e não em silos: enquanto aprende a criar tabelas e consultá-las em SQL, também está aprendendo a manipular vetores com NumPy e a produzir gráficos com Matplotlib.

O ponto de partida é conceitual. A primeira aula com conteúdo substantivo trata de **Inteligência Artificial**, situando as definições de IA, Aprendizado de Máquina e Aprendizagem Profunda, percorrendo a história da área (com seus ciclos de hype e de "inverno") e apresentando o fluxo típico de um projeto de Machine Learning, junto de uma lista de equívocos comuns cometidos por equipes de dados. Esse enquadramento é importante porque um sistema de apoio à decisão raramente é um algoritmo isolado: é um encadeamento de coleta, armazenamento, transformação, análise e apresentação de dados.

O bloco de **Python** cobre desde os conceitos mais elementares (comentários, indentação, variáveis, tipos de dados, operadores) até estruturas condicionais e de repetição, declaração de funções, importação de pacotes, computação vetorizada com NumPy, construção de gráficos com Matplotlib e Seaborn e, por fim, análise exploratória de dados. O bloco de **SQL** parte dos conceitos de chave e modelagem (conceitual, lógica e física), avança pela DDL (`CREATE`, `ALTER`, `DROP`), pela DML (`INSERT`, `UPDATE`, `DELETE`) e culmina na DQL, com `SELECT`, filtros, junções, funções de agregação, agrupamentos e subconsultas. O bloco de **Business Intelligence** apresenta o lugar dos painéis no processo de BI, os principais players de mercado, o Power BI e sua linguagem DAX, o tratamento de dados no Power Query, hierarquias, medidas e a construção de dashboards completos em oficinas práticas.

Vale notar que várias aulas da ementa correspondem a continuações ou aprofundamentos de um mesmo material de slides. As aulas 5, 6, 7 e 9, por exemplo, compartilham o conjunto de slides de SQL do Prof. Anderson Nascimento, avançando progressivamente nos tópicos; as aulas 4, 8 e 10 partilham os slides de introdução a Python e NumPy da Profa. Manoela Kohler; e as aulas 11, 13 e 15 apoiam-se no material de visualização de dados da Profa. Amanda Lemette. Esta apostila organiza o conteúdo respeitando a ordem da ementa, explicitando em cada aula o recorte específico que lhe cabe.

2 *Orientações Gerais de SAD

Material de slides não disponível para esta aula.

Esta primeira sessão é dedicada à apresentação da disciplina: sua ementa, a forma de avaliação, o calendário e a articulação entre os módulos de Inteligência Artificial, Python, SQL e Business Intelligence. Do ponto de vista de estudo, o que importa reter é a lógica de encadeamento do curso, descrita na visão geral acima: **dado bruto, armazenamento estruturado, transformação, análise e apresentação**. Cada aula subsequente ocupa um ponto específico dessa cadeia.

3 *Instalação de Softwares

A aula de instalação prepara o ambiente de trabalho. O material indica explicitamente a ferramenta **RapidMiner**, com duas alternativas de obtenção:

- A **versão utilizada no curso**, disponibilizada em pasta do Drive da disciplina;
- A **nova versão**, obtida na área de downloads da conta RapidMiner (`my.rapidminer.com`, seção de downloads).

Além do RapidMiner, o material de apoio inclui um vídeo demonstrando a instalação passo a passo. Ao longo do curso, outras ferramentas serão instaladas conforme a necessidade de cada bloco: o **PostgreSQL** com a IDE **pgAdmin 4** para o módulo de SQL, o **Power BI Desktop** para o módulo de visualização, e ambientes Python como **Jupyter**, **Spyder**, **PyCharm**, **VS Code** ou o **Google Colab** (que dispensa instalação local).

Uma orientação prática recorrente nos slides: a melhor IDE é aquela com a qual o usuário se sente mais à vontade. Não há necessidade de padronizar a ferramenta desde o primeiro dia; o essencial é ter um ambiente estável no qual o código executa.

4 Introdução à Inteligência Artificial

Esta aula estabelece o vocabulário conceitual da disciplina. Os objetivos de aprendizagem declarados nos slides são: definir Inteligência Artificial e seus subcampos, explicar como o Deep Learning supera limitações clássicas do Machine Learning, percorrer os desenvolvimentos históricos e os ciclos de inverno e hype da IA, diferenciar IA moderna de IA clássica, compreender possibilidades de aplicação, entender o fluxo de um projeto de ML e reconhecer equívocos comuns na área. O material tem como fonte parcial o curso *Artificial Intelligence 501*, formulado pela Intel.

4.1 Definições básicas

O material organiza três camadas encaixadas de definições:

- **Inteligência Artificial (IA)**: o campo mais amplo, que abriga as demais.
- **Aprendizado de Máquina (Machine Learning)**: conjunto de algoritmos que **aprendem ao ver repetidamente um conjunto de dados**, em vez de serem explicitamente programados por seres humanos. Essa é a diferença essencial em relação à programação convencional: não se codifica a regra, codifica-se o procedimento que descobre a regra.
- **Aprendizagem Profunda (Deep Learning)**: subcampo do ML que trabalha com **diferentes níveis de abstração**, extraindo hierarquicamente representações cada vez mais complexas a partir dos dados brutos.

Uma citação de Andrew Ng, da Universidade de Stanford, sintetiza a ambição do campo: cerca de cem anos atrás a eletricidade transformou todas as grandes indústrias, e a IA avançou até o ponto em que tem o poder de transformar todos os setores importantes nos próximos anos.

4.2 Terminologia e o problema das características

O exemplo canônico apresentado é a **classificação de espécies de flores**, base clássica para introduzir os termos de ML. Um segundo exemplo, mais próximo do mundo dos negócios, é a **detecção de transações fraudulentas de cartão de crédito**. Nesse caso, o analista precisa identificar as características (atributos) que permitirão ao algoritmo aprender quais combinações sugerem atividade incomum:

- Hora da transação;
- Valor da transação;
- Localização;
- Categoria da compra;
- entre outras.

Aqui reside a **limitação clássica do Machine Learning**: alguém precisa decidir quais características usar. Os slides propõem então uma pergunta provocativa: para determinar se uma imagem é de um gato ou de um cão, que características você usaria? Não há resposta trivial em termos de atributos tabulares. **É aí que entra o Deep Learning**, capaz de aprender as próprias representações a partir dos pixels, em níveis sucessivos de abstração.

4.3 História: dos primórdios aos invernos da IA

O material percorre a cronologia da área:

Early AI (a partir de 1950)

- **1950**: Alan Turing desenvolve o teste de Turing, para avaliar a capacidade das máquinas de apresentar comportamento inteligente.
- **1956**: a Inteligência Artificial é aceita como campo na Conferência de Dartmouth.
- **1957**: Frank Rosenblatt inventa o algoritmo **perceptron**, precursor das redes neurais modernas.
- **1959**: Arthur Samuel publica um algoritmo para um programa de damas usando aprendizado de máquina.

O primeiro inverno da IA

- **1966**: o comitê da ALPAC (presidido por John R. Pierce) avalia técnicas de IA para tradução de máquina e determina que o retorno não compensou o investimento.
- **A partir de 1969**: Marvin Minsky publica um livro sobre as limitações do perceptron, o que retarda a pesquisa em redes neurais.
- **1973**: o relatório Lighthill destaca o fracasso da IA.

Os dois relatórios levaram a cortes no financiamento governamental da pesquisa, configurando o primeiro **A.I. Winter**.

Boom dos anos 1980

- Surgem os **Sistemas Especialistas**, sistemas com regras programadas concebidas para imitar especialistas humanos, rodando em mainframes com linguagens especializadas (por exemplo, LISP).

- Foram as primeiras tecnologias de IA amplamente utilizadas: no pico, dois terços das empresas da *Fortune 500* usavam sistemas especialistas.
- **1986**: o algoritmo **backpropagation** viabiliza o treinamento de perceptrons de múltiplas camadas, renovando o interesse em redes neurais artificiais.

Outro inverno (final dos anos 1980 e início dos 1990)

- O progresso dos sistemas especialistas na resolução de problemas de negócio diminuiu.
- Esses sistemas passaram a ser fundidos em aplicativos de negócio gerais (SAP, Oracle), que rodavam em PCs em vez de mainframes.
- As redes neurais não escalaram para problemas maiores e mais complexos.
- O interesse empresarial em IA arrefeceu.

Aprendizado de Máquina Clássico (final dos anos 1990 e início dos 2000)

- Avanços no algoritmo **SVM** o tornam o método de escolha.
- Soluções em IA obtêm sucesso em reconhecimento de voz, diagnóstico médico e robótica.
- Algoritmos de IA são integrados em sistemas maiores e passam a ser úteis em toda a indústria.
- O **Deep Blue** derrota o campeão mundial de xadrez, Garry Kasparov.
- O motor de busca do Google é lançado usando inteligência artificial.

A ascensão do Deep Learning (a partir de 2006)

- **2006**: Geoffrey Hinton publica artigo sobre pré-treinamento não supervisionado, que permite treinar redes neurais mais profundas. O foco de estudo migra para a aprendizagem profunda.
- **2009**: a base **ImageNet**, de imagens rotuladas por humanos, é apresentada na conferência CVPR.
- **2010**: algoritmos competem em tarefas de reconhecimento visual na primeira competição ImageNet.

IA Moderna (2012 até o presente)

- **2012**: o Deep Learning bate os benchmarks anteriores na competição ImageNet.
- **2013**: o Deep Learning é usado para entender o significado conceitual de palavras.
- **2014**: avanços semelhantes surgem na área de tradução, com desdobramentos em pesquisa na web, pesquisa em documentos, resumo de documentos e tradução automática. No mesmo ano, algoritmos de visão computacional passam a descrever fotos.
- **2015**: o framework **TensorFlow** é desenvolvido.
- **2016**: o **AlphaGo**, da DeepMind, desenvolvido por Aja Huang, derrota Lee Se-dol, o maior campeão de Go.
- Em março, a **xAI** é fundada por Elon Musk.
- **2024 em diante**: a agenda passa a ser dominada por **LLM**, **SLM**, **VLM**, **modelos multimodais**, **Foundation Models**, além das discussões de **ética** e **regulamentação**.

4.4 Aplicações e fluxo de trabalho

O material aponta aplicações em **transporte**, **mídias sociais** e no **dia a dia**, e apresenta o fluxograma típico de um projeto de Machine Learning: um ciclo que vai da definição do problema à coleta e preparação dos dados, à modelagem, à avaliação e à colocação em produção.

4.5 Seis equívocos comuns

Esta é uma das partes mais úteis da aula para quem vai atuar profissionalmente com dados:

1. **O mito do unicórnio**: cientistas de dados especialistas em todas as áreas são chamados de unicórnios. Na prática, equipes bem-sucedidas contêm pessoas de diversas formações e habilidades: alguns se distinguem em comunicação, outros em estatística. As boas equipes reúnem especialistas em três áreas principais: **negócios, ciência e engenharia**.
2. **Foco exclusivo em pesquisa e algoritmos**: equipes de ciência de dados não podem se concentrar apenas nisso. Boas equipes conseguem identificar problemas, comunicar suas descobertas e trabalhar com a engenharia para achar o melhor método de produção.
3. **A solução mais complicada nem sempre é a melhor**: equipes tendem a ter mais êxito quando começam do básico e só então avançam para técnicas mais complexas. Modelos complexos podem ser mais precisos, mas são menos interpretáveis, mais propensos a falhar de forma imprevisível e mais difíceis de sustentar. Começar do básico também garante alinhamento com as necessidades do negócio.
4. **Técnicas não são universalmente transponíveis entre indústrias**: é necessário **conhecimento do domínio** para entender quais dados e problemas são relevantes. As técnicas de limpeza, armazenamento, extração de insights e modelagem são similares, mas a interpretação não é.
5. **Projetos não começam bem definidos**: projetos de ciência de dados são normalmente exploratórios e experimentais. Em geral não é claro quão difícil será a solução até que se explorem os dados. Gerentes de produto devem trabalhar com a equipe para gerenciar expectativas.
6. **Os melhores modelos não são necessariamente os mais avançados**: há mais desafios na escolha de um modelo do que a sua capacidade de previsão. Alguns modelos podem ser lentos ou complicados demais para a produção; outros podem não ser interpretáveis, o que torna investidores relutantes.

5 Introdução ao Python

Esta aula abre o bloco de programação. O programa declarado nos slides divide-se em duas partes: uma introdução geral (filosofia da linguagem, ferramentas para análise de dados, características, IDEs) e os conceitos básicos propriamente ditos (variáveis, tipos de dados, operações, estruturas condicionais e de repetição, bibliotecas e funções).

5.1 Lógica, algoritmos e programação

O ponto de partida é filosófico. A lógica é a parte da filosofia que trata das formas do pensamento em geral e das operações intelectuais que visam determinar o que é verdadeiro ou não. Transposta para a tecnologia da informação, a lógica é a **organização e o planejamento das instruções e assertivas em um algoritmo**, a fim de viabilizar a implantação de um programa. Programar, portanto, nada mais é do que a organização coesa de uma sequência de instruções voltadas à resolução de um problema de forma lógica.

5.2 Filosofia e características do Python

O material lista os atributos que explicam a popularidade da linguagem:

- Totalmente gratuito e **open-source**;
- Usabilidade;
- Orientado a objetos;
- Poderoso;
- Ferramentas gráficas poderosas;
- Flexível;
- Vasta comunidade;
- Compatibilidade;
- Usado mundialmente pela academia e pela indústria;
- Sempre atualizado, com novas bibliotecas de uso livre.

Entre as IDEs citadas estão **Jupyter**, **Spyder**, **PyCharm** e **VS Code**, além do **Google Colab**, ambiente em nuvem que dispensa instalação. A recomendação é pragmática: a melhor IDE é aquela em que você se sente mais à vontade.

5.3 Conceitos básicos

Comentários. Quando o programa cresce, fica difícil ler e manter. Por isso é boa prática inserir documentação ou notas no código, chamadas de comentários. Em Python, o comentário começa com o sinal de hash (`#`) e continua até o fim da linha; o interpretador ignora comentários. Há três usos típicos: comentário em linha, comentário em bloco e comentário de documentação (docstring) para funções, módulos, pacotes e classes.

Indentação. Python usa indentação para delimitar blocos, em vez de chaves. Tabulações e espaços são suportados.

```
# comentario em linha
if 10 > 5:
    print("dez e maior que cinco")  # bloco definido pela indentacao
```

Tipos de dados. O material organiza os tipos assim:

- **Numérico**: representa dados com valor numérico. `int` contém números inteiros positivos ou negativos, sem fração ou decimal; `float` é um número real com representação de ponto flutuante, especificado por um ponto decimal; e a classe `complex` representa números complexos, na forma (parte real) mais (parte imaginária) seguida de j.
- **Booleano**: tipo com um dos dois valores internos, verdadeiro ou falso, indicado pela classe `bool`.
- **Sequências**: coleções ordenadas de tipos de dados semelhantes ou diferentes, que permitem armazenar vários valores de forma organizada e eficiente. Há três tipos:
  - **String** (`str`): matrizes de bytes que representam caracteres; uma coleção de um ou mais caracteres entre aspas simples, duplas ou triplas. É possível acessar elementos individuais dentro de uma string por índice.

  - **Lista** (`list`): semelhante às matrizes de outras linguagens, mas não precisa ser homogênea, podendo conter inteiros, strings e objetos na mesma estrutura. É **mutável** e ordenada, com contagem definida.
  - **Tupla** (`tuple`): coleção ordenada de objetos, semelhante à lista, indexada por inteiros, mas **imutável**. Essa é a diferença importante entre lista e tupla.

- **Set**: coleção **não ordenada**, iterável, mutável e **sem elementos duplicados**. A ordem dos elementos é indefinida. A principal vantagem sobre a lista é possuir um método altamente otimizado para verificar se um elemento específico está contido no conjunto.
- **Dicionário** (`dict`): coleção não ordenada de valores de dados usada para armazenar dados como um mapa. Diferentemente de outros tipos, que mantêm um valor único como elemento, o dicionário mantém um par **chave:valor**. Cada chave é separada de seu valor por dois pontos, e cada par é separado do seguinte por vírgula.

```
inteiro = 6
decimal = 2.5
verdade = True
texto = "The Shining"
lista = [4.5, 4.0, 5.0]
tupla = (1, 2, 3)
conjunto = {1, 2, 3}
dicionario = {"nome": "Ana", "sexo": "F"}
```

Operadores e type casting. Os slides apresentam a tabela de operadores (aritméticos, de comparação, lógicos) e o conceito de **type casting**, a conversão explícita entre tipos.

Interação com o usuário. A leitura de dados do teclado permite escrever scripts interativos. Um exercício proposto pede um script que pergunte quantos anos o usuário tem, aguarde a resposta e então imprima a idade, o ano de nascimento (considerando que a pessoa já fez aniversário no ano corrente) e o tipo da variável.

5.4 Estruturas condicionais e de repetição

O material cobre as estruturas condicionais e os dois laços fundamentais, o **while loop** e o **for loop**.

```
numeros = [12, 23, 11, 34, 13, 56, 102, 101, 13]
for n in numeros:
    if n % 2 == 0:      # resto da divisao por 2
        print(n)
```

5.5 Exercícios propostos

Os exercícios da aula constroem progressivamente a fluência na linguagem:

1. Criar uma variável com valor 6 (inteiro) e outra com valor 2 (inteiro), dividir a primeira pela segunda salvando em nova variável e verificar o tipo do resultado da divisão. O ponto pedagógico é que a divisão em Python produz `float`, mesmo entre inteiros.

2. Dois amigos dividem o lucro obtido por meio de um site de consultoria em projetos de IA. O lucro mensal é de 8 mil; o amigo 1 tem direito a 30% e o restante pertence ao amigo 2. Calcular o lucro total do ano e o de cada amigo.
3. Prever de cabeça o resultado da expressão `5 + 3 * 10 / 3 == 15` e confirmar no Python. O exercício testa a compreensão da precedência de operadores.

Um segundo conjunto trabalha com listas, a partir de dados sobre o filme *The Shining*:

```
movieName = "The Shining"
actors = ["Jack Nicholson", "Shelley Duvall", "Danny Lloyd",
          "Scatman Crothers", "Barry Nelson"]
scores = [4.5, 4.0, 5.0]
comments = ["Best Horror Film I Have Ever Seen",
            "A truly brilliant and scary film from Stanley Kubrick",
            "A masterpiece of psychological horror"]
```

As tarefas são: criar uma lista com os quatro elementos carregados (nome do filme, atores, avaliações e comentários) e resolver tudo a partir dessa lista; imprimir a string com o nome do primeiro ator; e imprimir a melhor avaliação do filme, com score e comentário. As dicas fornecidas são `max(lista)`, que retorna o valor máximo, e `lista.index(valor)`, que retorna o índice do valor na lista.

Um terceiro conjunto explora repetição e aleatoriedade: imprimir uma sequência de 25 números consecutivos; imprimir essa sequência duas vezes (com laço aninhado); filtrar os números pares de uma lista; simular um sorteio da Mega Sena, com 6 números entre 1 e 60, com e sem estrutura de repetição, usando `np.random.randint()` e `np.random.choice()`; e simular o resultado de um dado de 6 faces jogado 7 vezes. A pergunta reflexiva proposta é pertinente: uma estrutura de repetição com `np.random.randint` sem nenhum tratamento pode gerar algum problema? Sim, pode gerar números repetidos no mesmo sorteio, o que a função `np.random.choice()` com amostragem sem reposição resolve.

Por fim, pede-se um script que calcule o **fatorial** de um número qualquer, lembrando que 5! = 5 x 4 x 3 x 2 x 1 = 120.

5.6 Lei dos Grandes Números

Um exercício de destaque é a verificação empírica da **Lei dos Grandes Números**, segundo a qual a média amostral converge para o valor esperado quando o número de experimentos cresce: $X_n \to E(X)$ quando $n \to \infty$.

Os slides mostram a convergência numa simulação binária:

- 10 experimentos: 7 contra 3, ou seja, 70% e 30%;
- 100 experimentos: 52 contra 48, ou seja, 52% e 48%;
- 1000 experimentos: 502 contra 498, ou seja, 50,2% e 49,8%.

A tarefa seguinte generaliza: testar a lei para N números aleatórios gerados com distribuição normal de média 0 e desvio padrão 1; criar um script que conte quantos desses números caem entre -1 e 1 e divida por N. Sabe-se que o valor esperado é de 68,2%. Deve-se verificar que a média se aproxima desse valor conforme N cresce.

```
import numpy as np
amostra = np.random.normal(size=1)
```

5.7 Funções

Funções são o primeiro passo para a **reutilização de código**: permitem definir um bloco reutilizável que pode ser usado repetidamente no programa. O Python fornece funções internas como `print()` e `len()`, mas o usuário também pode definir as suas. A anatomia apresentada nos slides é: **parâmetros de entrada**, corpo da função e **retorno**. Uma regra prática enfatizada: a função deve ser declarada antes de sua chamada.

Os exercícios sobre funções pedem:

1. Criar uma função que calcule o fatorial de um número qualquer;
2. Usar a função `factorial` do pacote `math` para o mesmo fim;
3. Criar uma função que receba o raio de um círculo e retorne sua área, usando `math.pi` e o operador de potência `**`;
4. Alterar essa função para retornar também o diâmetro e o perímetro, e em seguida definir um valor padrão de 5 para o raio;
5. Criar uma função que valide a Lei dos Grandes Números, recebendo o número de experimentos como parâmetro e, depois, um parâmetro opcional de intervalo com valores padrão -1 e 1, para que se possa testar outros intervalos.

```
import math

def circulo(raio=5):
    area = math.pi * raio ** 2
    diametro = 2 * raio
    perimetro = 2 * math.pi * raio
    return area, diametro, perimetro
```

6 Linguagem de definição de Dados

Aqui começa o bloco de SQL, conduzido pelo Prof. Anderson Nascimento. A ementa do módulo abrange introdução à linguagem SQL, DDL, DML, DQL, Joins, Subconsultas, Funções e Views. O objetivo declarado é apresentar os principais comandos de criação de objetos e recuperação de dados em bancos relacionais.

6.1 Competências e ferramentas

As competências visadas são: conhecer a linguagem SQL; criar objetos com comandos DDL; manter dados com comandos DML; praticar consultas com comandos DQL; realizar junções com Inner Join e Equi Join; criar consultas avançadas com subconsultas; aplicar funções em consultas; e criar visões de dados.

As ferramentas indicadas são o SGBD **PostgreSQL**; para modelagem, **Astah Community**, **BR Modelo** e **SQL Power Architect**; e, para prática online, os sites `sqliteonline.com`, o simulador

SQL da W3Schools e o `sqlfiddle.com`. A bibliografia de referência é *SQL Structured Query Language*, de Luís Damas (6ª edição, LTC, 2007) e *Sistemas de Banco de Dados*, de Ramez Elmasri e Shamkant Navathe (7ª edição, Pearson, 2018).

6.2 Conceitos iniciais

A **SQL (Structured Query Language)** é a linguagem padrão de consultas aos bancos de dados relacionais. Seus comandos podem ser agrupados em cinco categorias:

- **DDL** — Linguagem de Definição de Dados;
- **DML** — Linguagem de Manipulação de Dados;
- **DCL** — Linguagem de Controle de Dados;
- **DTL** — Linguagem de Transação de Dados;
- **DQL** — Linguagem de Consultas de Dados.

6.3 Conceito de chaves

O conceito de chaves é fundamental para garantir a consistência dos dados e evitar redundância, isto é, repetição de dados nas tabelas.

- **Chave primária**: identificador único, que não pode ser vazio nem se repetir. Deve ser um dado capaz de identificar uma linha em uma tabela; por isso seu uso é obrigatório.
- **Chave candidata**: campo elegível para ser chave primária, ou seja, que também não pode se repetir nem ser vazio. Normalmente é declarado como único na criação da tabela.
- **Chave estrangeira**: campo que faz referência a outro campo chave, localizado em outra tabela.

6.4 Um pouco de modelagem

Os modelos de banco de dados dividem-se em três categorias:

- **Conceitual**: visão de alto nível, sem implementação, onde se define **o que** deverá ser feito.
- **Lógico**: visão próxima da implementação das tabelas, mostrando como elas se relacionam, já com chave primária e chave estrangeira.
- **Físico**: a implementação em si, isto é, os dados no SGBD.

O modelo conceitual pode ser representado por meio do **Diagrama de Classes da UML** ou do **Diagrama Entidade Relacionamento (DER)**. Ferramentas sugeridas: Astah, Dia, Lucidchart e Draw.io para UML; BrModelo (versões 2.0, 3.2 e Web) para DER.

Cardinalidade e participação. A multiplicidade é a composição de **participação** (opcional ou obrigatória, isto é, 0 ou 1, indicando o número mínimo de ocorrências) e **cardinalidade** (número máximo de ocorrências, sendo N indefinido). A notação DER usa parênteses, como (0,n); a UML usa dois pontos e asterisco, como `0..*`. Assim:

Notação DER	Notação UML	Significado
(1,1)	1..1	Obrigatório

Notação DER	Notação UML	Significado
(0,1)	0..1	Participação opcional
(0,N)	`0..*`	Cardinalidade indefinida

Regras de mapeamento relacional. São as regras que traduzem o modelo conceitual em tabelas:

- **(1,1) com (1,1)**, obrigatório dos dois lados: caso raro; normalmente há fusão de tabelas, ou escolhe-se o lado em que a chave estrangeira vai ficar.
- **(0,1) com (0,1)**, opcional dos dois lados: caso raro; escolhe-se o lado em que a chave estrangeira ficará.
- **(0,1) com (1,1)**, opcional de um lado e obrigatório de outro: caso comum; o lado obrigatório cede a chave estrangeira.
- **(0,N) com (0,N)**, muitos em ambos os lados, chamado relacionamento M:N: exige a criação de uma nova tabela. São equivalentes os casos (1,N) com (0,N), (0,N) com (1,N) e (1,N) com (1,N).
- **(0,1) com (0,N)**, cardinalidade 1 de um lado e N de outro: caso comum; o lado com cardinalidade 1 cede uma chave estrangeira ao lado com cardinalidade N.

As mesmas regras valem em notação UML, substituindo (1,1) por 1..1, (0,1) por 0..1 e (0,N) por `0..*`.

6.5 O SGBD PostgreSQL

O PostgreSQL é o banco de dados utilizado no curso. Trata-se de um dos bancos livres mais poderosos do mercado e um dos mais utilizados no mundo, conforme o ranking de SGBDs do site `db-engines.com` consultado em janeiro de 2025. A versão indicada nos slides é a **17.2**, que requer a IDE **pgAdmin 4**; para Windows 32 bits, a última versão é a 10. Possui versão para Mac e pode ser baixado no site da EnterpriseDB.

6.6 Linguagem DDL

A DDL apresenta os comandos de criação, alteração e eliminação de objetos em um banco de dados. Os comandos são **CREATE**, **ALTER** e **DROP**.

CREATE. Utilizado para criar objetos como tabelas, índices, visões e esquemas. A sintaxe básica para criação de tabelas:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo primary key,
  coluna2 tipo [restricao],
  coluna_n tipo [restricao]);
```

Os principais tipos de dados no PostgreSQL, segundo o material:

- `int` ou `integer`, `bigint`, `serial`, `smallint` — inteiros;
- `real`, `float`, `money`, `decimal(x,y)` — números decimais;
- `varchar(n)` — texto de tamanho variável, em que n é o número máximo de caracteres;

- `char(n)` — texto de tamanho fixo, em que n é o número definido de caracteres;
- `date` — datas;
- `timestamp` — carimbo de data e hora.

Chaves estrangeiras. Viabilizam o relacionamento entre tabelas, minimizando a repetição de dados. A sintaxe na criação:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo primary key,
  coluna2 tipo [restricao],
  coluna_n tipo [restricao],
  foreign key (coluna_n) references tabela_referenciada (chave_primaria)
);
```

ALTER. Usado sempre que for preciso modificar uma tabela. As dez formas apresentadas nos slides:

```
ALTER TABLE t ADD COLUMN nome_coluna TIPO;                -- 1) nova coluna
ALTER TABLE t ALTER COLUMN nome_coluna TYPE novo_tipo;    -- 2) alterar tipo
ALTER TABLE t DROP COLUMN nome_coluna;                    -- 3) apagar coluna
ALTER TABLE t ALTER COLUMN nome_coluna SET DEFAULT valor; -- 4) valor default
ALTER TABLE t RENAME COLUMN nome_antigo TO nome_novo;     -- 5) renomear coluna
ALTER TABLE t RENAME TO nome_novo;                        -- 6) renomear tabela
ALTER TABLE t ALTER COLUMN c SET NOT NULL;                -- 7) alterar restricao
ALTER TABLE t ALTER COLUMN c DROP NOT NULL;               -- 8) remover restricao
ALTER TABLE t ADD PRIMARY KEY (c);                        -- 9) chave primaria
ALTER TABLE t ADD FOREIGN KEY (c)
  REFERENCES tabela_referenciada (coluna_referenciada);   -- 10) chave estrangeira
```

DROP. Elimina um objeto do banco de dados:

```
DROP TABLE nome_da_tabela;           -- apaga uma tabela
DROP TABLE nome_da_tabela CASCADE;   -- apaga tabela com dependencias
DROP TABLE tabela1, tabela2;         -- apaga mais de uma tabela
```

6.7 Exercícios de DDL

A sequência de exercícios constrói o banco `escola`: criar o database e a tabela `aluno` (exercício 1); criar a tabela `departamento` (2); criar a tabela `curso` (3); criar a tabela `matricula` (4). Em seguida, uma bateria sobre `ALTER`: criar a tabela `func` (5); renomear o campo `city` para `cidade` (6); criar a coluna `sexo` do tipo `char(1)` (7); alterar o tipo de `nomefunc` para `varchar(30)` (8); apagar a coluna `CBO` (9); definir `UF` com valor padrão RJ (10); renomear a tabela para `funcionario` (11); definir `nomefunc` como `not null` (12); e definir `coddepto` da tabela funcionário como chave estrangeira apontando para `coddepto` de departamento (13).

Os exercícios 14 a 16 demonstram na prática a integridade referencial: criar uma tabela `telefone`, criar uma chave estrangeira no campo `codfunc` apontando para funcionário, e então tentar apagar a tabela `funcionario` sem o `CASCADE`. Não funciona, exatamente porque existe dependência. É preciso apagar antes a tabela `telefone`. Uma nota importante dos slides: no modelo conceitual não representamos as chaves estrangeiras; apenas no modelo lógico é que fazemos tal representação.

6.8 Desafio 1

O desafio consolida vários conceitos de uma vez. Dadas as especificações, deve-se implementar o modelo em SQL:

- `codcliente`, chave primária, com numeração automática (o que sugere o tipo `serial`);
- O campo `sexo` deve permitir M ou F, e pode ser vazio;
- O campo `estcivil` deve permitir apenas Solteiro, Casado, Separado, Divorciado ou Viúvo, e não pode ser vazio;
- O campo `UF` deve armazenar o valor padrão RJ, ou seja, caso a UF não seja digitada, o banco gravará RJ;
- O campo `datacadastro` deve gravar automaticamente a data e a hora do cadastro;
- Devem ser armazenados 5 clientes, todos do Rio de Janeiro;
- O código do `INSERT` deve informar apenas nome, sexo e estcivil; todos os demais campos devem ser registrados automaticamente pelo SGBD.

Há ainda um **exercício extra**: implementar em SQL o banco de dados de um modelo transacional de sistema de controle de multas em rodovias, que será a base usada nas aulas seguintes por meio do script `multas.sql`.

7 Linguagem de Manipulação de Dados

A **DML** é a parte da SQL responsável por manusear os dados nas tabelas. Além dos comandos de recuperação baseados no `SELECT` (que pertencem à DQL), há três comandos importantes de manipulação: **INSERT**, **UPDATE** e **DELETE**.

7.1 INSERT

Responsável por inserir dados em uma tabela. A sintaxe é simples:

```
INSERT INTO nome_da_tabela (coluna1, coluna2, coluna_n)
VALUES (valor1, valor2, valor_n);
```

A inserção também pode ser feita com vários registros de uma só vez, separando as tuplas por vírgula. E, quando se preenchem todas as colunas sem pular nenhuma, é possível omitir o nome dos campos:

```
INSERT INTO nome_da_tabela VALUES
  (valor1, valor2, valor_n),
  (valor1, valor2, valor_n);
```

Uma advertência prática dos slides: atenção aos tipos de dados. Os textos devem ser inseridos com **aspas simples** e as datas no **formato americano** (ano, mês, dia).

7.2 Exercícios de INSERT

O exercício 17 pede inserir na tabela `aluno` o registro `1, Ana, F, 1979-01-23`. O 18 pede os demais alunos: André, Andreia, Bruna e Bruno, com seus respectivos sexos e datas de nascimento. O 19 cadastra os departamentos `1, TI` e `2, ADM`. O exercício 20 cadastra os cursos:

```
INSERT INTO curso VALUES
  (1, 'Python', 40, 1200, 1),
  (2, 'Power BI', 20, 900, 1),
  (3, 'Pentaho', 40, 1200, 1),
  (4, 'Recursos Humanos', 60, 2000, 2),
  (5, 'Marketing', 80, 2500, 1);
```

O exercício 21 pede matricular os alunos nos cursos: Python em 2019-04-01; Power BI em 2019-04-02; Pentaho em 2019-04-03; Recursos Humanos em 2019-04-04; Marketing em 2019-04-06. Os slides trazem a solução, com dezesseis tuplas de (código do aluno, código do curso, data). O exercício 22 cadastra três funcionários: Paulo, de Duque de Caxias/RJ, departamento 1, sexo M; Paula, do Rio de Janeiro/RJ, departamento 2, sexo F; e José, do Rio de Janeiro/RJ, departamento 1, sexo M.

7.3 UPDATE

Permite a alteração dos dados já inseridos. É preciso especificar a cláusula `WHERE` para evitar que **todos** os registros sejam modificados erroneamente:

```
UPDATE nome_da_tabela SET nome_do_campo = alteracao_desejada
WHERE condicao_para_alteracao;

-- exemplo: alterar a cidade do funcionario Paulo
UPDATE funcionario SET cidade = 'Rio de Janeiro'
WHERE nomefunc = 'Paulo';
```

Os exercícios 23 a 28 cobrem casos progressivamente mais elaborados: alterar o preço do curso de Power BI para 1000; mover o curso de Marketing para o departamento ADM (2); alterar a carga horária de Recursos Humanos para 80; dar 20% de aumento nos cursos da área de TI; conceder 50 reais de desconto nos cursos da área de ADM; e conceder 10% de desconto aos cursos com carga horária menor que 80 cujo valor seja maior que 1000 reais. Note que os três últimos exigem operações aritméticas sobre a própria coluna e condições compostas no `WHERE`.

7.4 DELETE

Elimina linhas de uma tabela. A distinção conceitual é importante: **não confunda DELETE com DROP**. O `DELETE` mantém a estrutura da tabela, eliminando apenas os dados; o `DROP` apaga toda a estrutura.

```
DELETE FROM nome_da_tabela;              -- apaga todas as linhas
DELETE FROM nome_da_tabela WHERE condicao;  -- apaga linhas filtradas
```

Os exercícios 29 e 30 pedem apagar todos os dados da tabela funcionário e deletar as matrículas realizadas após o dia 4 de abril pela aluna Andreia.

8 Linguagem de Consultas

A **DQL** é responsável por produzir consultas na base de dados. Apesar de não possuir muitos comandos específicos, há diversas cláusulas que podem ser combinadas para chegar aos resultados esperados. Basicamente, trabalha-se com a cláusula `SELECT`.

Antes de iniciar, o material orienta criar no PostgreSQL um novo database chamado `multas` e rodar o script `multas.sql`, cedido pelo professor. É essa base, com tabelas de proprietário, carro, ocorrência e rodovia, que serve de laboratório para todos os exercícios seguintes.

8.1 O comando SELECT

O `SELECT` é usado para selecionar as colunas esperadas como resposta do banco. Pode-se usar o asterisco para ver todas as colunas:

```
SELECT coluna1, coluna2, coluna_n FROM nome_da_tabela;
SELECT * FROM nome_da_tabela;
```

8.2 ORDER BY e DISTINCT

A cláusula **ORDER BY** altera a forma como os resultados são exibidos, em ordem ascendente ou descendente. O `ASC` é opcional quando a ordem de exibição é crescente:

```
SELECT coluna1 FROM tabela ORDER BY coluna_ordenada ASC;
SELECT coluna1 FROM tabela ORDER BY coluna_ordenada DESC;
```

Quando os resultados se repetem desnecessariamente, usa-se **DISTINCT**, que faz o banco exibir resultados sem duplicatas:

```
SELECT DISTINCT coluna1 FROM tabela1;
```

8.3 WHERE e operadores de comparação

Os filtros limitam os resultados, permitindo que o banco retorne apenas o que satisfaz determinada necessidade. Múltiplas condições podem ser combinadas com **AND** e **OR**:

```
SELECT coluna1 FROM tabela1
WHERE condicao1 OR condicao2 AND condicao3;
```

Os operadores de comparação disponíveis são: maior, menor, maior ou igual, menor ou igual, e os dois operadores de desigualdade, escritos como `<>` ou `!=`.

8.4 LIKE

A cláusula **LIKE** permite buscar resultados filtrando apenas por uma parte do texto, usando o coringa `%`. A posição do símbolo determina o comportamento da busca:

```sql
SELECT nome FROM cliente WHERE nome LIKE '%MARIA%';  -- contem
SELECT nome FROM cliente WHERE nome LIKE '%MARIA';   -- termina com
SELECT nome FROM cliente WHERE nome LIKE 'MARIA%';   -- comeca com
```

O primeiro caso retornaria nomes como ANA MARIA, MARIA SANDRA, MARIANA SILVA e MARIAH DO NASCIMENTO. O segundo retornaria Ana Maria e Luana Silva Maria. O terceiro retornaria Maria Sandra e Mariana Silva.

8.5 Exercícios de DQL

Os exercícios 31 a 46 percorrem toda essa gramática sobre a base de multas. Alguns exemplos representativos:

- Recuperar o nome e o estado de cada proprietário (31) e todos os dados dos veículos cadastrados (32);
- Recuperar o nome dos proprietários em ordem alfabética (33);
- Recuperar todas as datas de multas da mais recente para a mais antiga (34), e depois a mesma consulta sem repetições (35);
- Recuperar apenas o nome dos proprietários de veículos do Rio de Janeiro (36) e, em seguida, o nome e o estado dos proprietários do Rio de Janeiro e de São Paulo (37);
- Recuperar nome e data de nascimento dos proprietários do Rio de Janeiro e de São Paulo nascidos antes de 1990, ordenados crescentemente por data de nascimento (38);
- Exibir o nome de todos os proprietários **exceto** os da região Sudeste (39);
- Exibir todos os fabricantes existentes, sem repetições e em ordem alfabética (40);
- Recuperar a lista de estados, em ordem alfabética e sem repetição, que possuem proprietários de veículos (41);
- Recuperar o código de todas as rodovias, sem repetição, onde houve ocorrências (42);
- Recuperar a placa dos carros, em ordem decrescente e sem repetição, que tiveram alguma ocorrência (43);
- Recuperar a descrição das multas relacionadas à CNH do motorista (44), o nome dos motoristas que começam com a letra A (45) e a placa dos veículos cuja placa termina com 0 (46). Estes três exigem `LIKE`.

9 Introdução ao Python - continuação

Esta aula retoma e aprofunda o material introdutório de Python, agora com foco na fixação por meio de exercícios e na transição para o trabalho com pacotes. O recorte cobre a segunda metade dos slides `Aulas01e02_Intro.pdf`: estruturas de repetição, funções e o exercício da Lei dos Grandes Números, além do início do material sobre pacotes.

9.1 Consolidando estruturas de repetição

Nesta etapa, os laços deixam de ser exercícios sintáticos e passam a ser instrumentos de simulação. É aqui que os exercícios de Mega Sena, de lançamento de dado e da Lei dos Grandes Números adquirem sentido pleno: eles são simulações estocásticas escritas com laços.

```python
import numpy as np

# um sorteio de 1 numero entre 1 e 60
np.random.randint(1, 61, 1)

# seis numeros distintos entre 1 e 60
np.random.choice(range(1, 61), 6, replace=False)
```

O ponto pedagógico é a comparação entre `np.random.randint()`, que sorteia com reposição e portanto pode repetir números no mesmo jogo, e `np.random.choice()` com `replace=False`, que garante unicidade. Trata-se de uma primeira lição sobre o cuidado com a semântica das funções de biblioteca.

9.2 Funções como unidade de reuso

A aula formaliza a declaração de funções. Uma função encapsula um bloco de código com parâmetros de entrada e um valor de retorno, e deve ser declarada antes de ser chamada. Os parâmetros podem ter **valores padrão**, o que permite chamadas mais curtas sem perder a flexibilidade.

```python
def lei_grandes_numeros(n, inferior=-1, superior=1):
    amostra = np.random.normal(size=n)
    dentro = ((amostra > inferior) & (amostra < superior)).sum()
    return dentro / n
```

Essa função sintetiza os dois exercícios finais: recebe o número de experimentos como parâmetro obrigatório e o intervalo a validar como parâmetro opcional, com valores padrão -1 e 1. Ao variar o intervalo, é possível verificar outras proporções da distribuição normal.

9.3 Pacotes

O material introduz as quatro formas canônicas de importação, que serão usadas em todo o restante do curso:

```python
import pacote                    # importa todos os modulos do pacote
from pacote import nome          # importa um modulo ou funcao especifico
from pacote import *             # importa tudo, chamavel sem o prefixo
from pacote import nome as apelido   # importa com nome alternativo
```

O exercício proposto integra os dois blocos: importar a classe `pyplot` do pacote `matplotlib`, que contém a função `hist()` capaz de gerar um histograma; usar as amostras de diferentes tamanhos geradas com `np.random` para criar um histograma de cada uma; e avaliar o gráfico gerado. Visualmente, o aluno reencontra a Lei dos Grandes Números: quanto maior a amostra, mais o histograma se aproxima da forma de sino da distribuição normal.

10 Joins e Funções

Esta aula fecha o bloco de SQL, tratando das junções entre tabelas, das funções de agregação, das consultas de agrupamento e das subconsultas.

10.1 Joins

Uma das principais características do modelo relacional é permitir a **junção entre tabelas**, por meio de chaves primárias e estrangeiras. Esse recurso é justamente o que ajuda a manter a consistência do banco e a eliminar redundâncias desnecessárias. Existem basicamente duas formas de realizar a junção: o **EQUIJOIN** e o **INNER JOIN**.

Equijoin. É bastante simples e ocorre igualando-se, na cláusula `WHERE`, as chaves primárias e estrangeiras das tabelas que se deseja relacionar:

```
SELECT tabela1.coluna1, tabela2.coluna1
FROM tabela1, tabela2
WHERE tabela1.pk = tabela2.fk;
```

Para evitar redigitar o nome das tabelas, pode-se dar apelidos a elas; o uso do `AS` é opcional:

```
SELECT t1.coluna1, t2.coluna1
FROM tabela1 AS t1, tabela2 AS t2
WHERE t1.pk = t2.fk;
```

Inner Join. Neste tipo de junção usa-se a palavra `JOIN` para realizar a união das tabelas. O uso da palavra `INNER` é opcional; é comum escrever apenas `JOIN`:

```
SELECT tabela1.coluna1, tabela2.coluna1
FROM tabela1 INNER JOIN tabela2
ON tabela1.pk = tabela2.fk;
```

Os exercícios 47 a 51 exercitam as duas formas: recuperar o nome dos proprietários em ordem alfabética junto com o nome do fabricante de seus carros; recuperar o nome dos proprietários, em ordem alfabética e sem repetição, que tiveram alguma ocorrência relatada; recuperar o nome do proprietário, o nome do fabricante do carro e a descrição da multa registrada na tabela ocorrência (essa última exige a junção de três tabelas); e, com `INNER JOIN`, recuperar os fabricantes de carros multados e os proprietários que tiveram alguma multa.

10.2 Funções de agregação

A linguagem SQL possui inúmeras funções que facilitam a recuperação de informações. Entre as principais:

- **SUM** — fornece o resultado da soma de uma coluna a partir de uma query;
- **COUNT** — conta a quantidade de linhas resultantes de uma query;
- **MAX** — recupera o valor máximo de uma coluna;
- **MIN** — recupera o valor mínimo de uma coluna;
- **AVG** — recupera a média de determinado valor.

Os exercícios 52 a 57 aplicam essas funções à base de multas: valor da multa mais cara; valor da multa mais barata; maior quantidade de pontos possíveis em uma única multa; valor médio das multas cadastradas; quantidade de multas com mais de 5 pontos; e valor total em multas relacionadas à CNH.

10.3 Consultas de agrupamento

É possível agrupar resultados combinando funções com a cláusula **GROUP BY**, para obter somatórios, contagens, médias e valores mínimos e máximos por categoria:

```
SELECT count(*), sexo
FROM funcionarios
GROUP BY sexo;
```

Neste caso, a query retorna o número de funcionários do sexo masculino e o número do sexo feminino. Os exercícios 58 a 60 aplicam o conceito: recuperar o número de proprietários por UF em ordem decrescente; recuperar a quantidade de carros por fabricante, exibindo apenas os três fabricantes com mais carros (usando a cláusula `LIMIT`); e recuperar o código da rodovia com mais ocorrências.

10.4 Subconsultas

Muitas vezes é preciso utilizar subconsultas para recuperar determinado resultado. Uma **subconsulta é uma consulta dentro de outra consulta**. O exemplo dos slides ilustra o padrão de agregação sobre agregação:

```
SELECT max(totmultas)
FROM (
  SELECT count(*) AS totmultas, placa
  FROM ocorrencia
  GROUP BY placa) AS r;
```

A consulta interna produz a contagem de multas por placa; a consulta externa extrai o valor máximo desse conjunto de contagens.

Outro padrão importante é o uso das cláusulas **IN** e **NOT IN**, aplicadas aqui para encontrar os proprietários que **não** tiveram multas:

```
SELECT nomeproprietario
FROM proprietario
WHERE nomeproprietario NOT IN (
  SELECT DISTINCT p.nomeproprietario
  FROM ocorrencia o, carro c, proprietario p
  WHERE o.placa = c.placa
    AND c.idproprietario = p.idproprietario
)
ORDER BY nomeproprietario;
```

Esse é um exemplo de raciocínio por complemento: primeiro constrói-se o conjunto dos que **tiveram** multas, unindo três tabelas por equijoin, e depois exclui-se esse conjunto do universo total. O exercício 61 pede aplicar a mesma lógica em outro nível de agregação: recuperar os estados que não possuem ocorrências de multas.

11 Numpy

O **NumPy** é o pacote básico fundamental da linguagem Python para computação científica. Ele provê muitas das estruturas básicas, fornecendo ferramentas para integração e comunicação com Fortran e C, vetorização, álgebra linear e geração de números aleatórios. O NumPy constitui uma fundação sólida para muitas ferramentas aplicadas em análise de dados.

O que o torna tão popular é sua **eficiência quando comparado às listas nativas**: é muito mais rápido trabalhar com NumPy, pois o pacote permite acesso a dados de modo muito mais eficiente.

11.1 Estruturas de dados

O material constrói a hierarquia de estruturas por dimensionalidade:

- **Vetores**: um vetor numérico em Python é uma sequência de elementos indexados de 0 a n-1, todos **de um mesmo tipo**.
- **Matrizes**: estruturas bidimensionais, organizadas em linhas e colunas.
- **Escalares, vetores, matrizes e tensores**: a generalização progressiva. Um escalar tem zero dimensões, um vetor tem uma, uma matriz tem duas e um tensor tem três ou mais.

11.2 Criação de arrays

```
import numpy as np

t1 = np.ones((4, 3, 2))            # tensor 3D preenchido com 1
t2 = np.zeros((4, 3, 2))           # tensor 3D preenchido com 0
t3 = np.random.random((4, 3, 2))   # valores aleatorios entre 0 e 1
t1.shape                           # (4, 3, 2)

X = np.array([[1, 2], [3, 4], [5, 6]])
X.shape                            # (3, 2)
v = np.array([1, 2, 3, 4, 5, 6])
```

O atributo `shape` informa o formato da estrutura e é o primeiro recurso de diagnóstico ao trabalhar com arrays. Os slides tratam ainda dos demais **atributos de um NumPy array** e das **funções para arrays**.

11.3 Operações

A distinção mais importante da aula é entre multiplicação elemento a elemento e multiplicação matricial. Dadas as matrizes A, com linhas (1, 2) e (4, 5), e B, com linhas (10, 20) e (30, 40):

```
A * B      # element-wise: linhas (10, 40) e (120, 200)
A.dot(B)   # multiplicacao matricial: linhas (70, 100) e (190, 280)
2 * A      # multiplicacao por escalar: linhas (2, 4) e (8, 10)
```

Confundir `*` com `.dot()` é uma das fontes mais comuns de erro silencioso em código numérico, porque ambas as operações produzem um resultado válido, apenas com significados diferentes.

11.4 Exercício de desempenho

Operações vetorizadas são extremamente mais eficientes do que a utilização de laços. O exercício proposto pede provar isso empiricamente: criar dois vetores com 100.000 elementos e fazer o produto escalar (dot product), comparando o tempo com o de listas nativas do Python. A dica é usar a função `time.process_time()` do pacote `time` para salvar o tempo antes e depois da operação; o tempo decorrido em segundos é a diferença entre esses valores.

```
import time

inicio = time.process_time()
resultado = a.dot(b)
decorrido = time.process_time() - inicio
```

11.5 Exercício: análise de demonstração financeira

Este é o exercício integrador da aula. O cenário: você é um cientista de dados de uma empresa de consultoria e um colega do departamento de auditoria pediu ajuda para avaliar o demonstrativo financeiro de uma organização X. Você recebeu dois vetores, com a receita mensal e a despesa mensal do ano:

```
receitas = np.array([14574.49, 7606.46, 18611.41, 19175.41, 8758.65,
                     8105.44, 11496.28, 9766.09, 10305.32, 18379.96,
                     10713.97, 15433.50])
custos = np.array([12051.82, 5695.07, 12319.20, 12089.72, 8658.57,
                   840.20, 3285.73, 5821.12, 6976.93, 16618.61,
                   10054.37, 3803.96])
```

As métricas financeiras a calcular são:

- O **lucro** de cada mês;
- O **lucro após imposto** de cada mês, sendo o imposto de 30% sobre o lucro;
- A **margem de lucro** de cada mês, ou seja, o lucro depois do imposto dividido pela receita;
- Os **meses bons**, aqueles em que o lucro após taxação foi maior que o lucro médio do ano, usando a função `np.where`;
- Os **meses ruins**, o contrário dos bons;
- O **melhor mês** e o **pior mês**, descontado o imposto.

As funções úteis indicadas são `mean()`, `max()`, `min()` e `np.where()`. O valor pedagógico do exercício está em resolver todas essas perguntas **sem escrever um único laço**: cada métrica é uma operação vetorizada sobre os dois arrays originais.

12 Declaração de funções

Esta aula se apoia no material de visualização de dados da Profa. Amanda Lemette (disciplina EQM2009, Python para Pesquisa Científica) e no material extra sobre publicação de bibliotecas Python no GitHub. O fio condutor é a passagem da função como bloco reutilizável dentro de um script para a função como componente de uma **biblioteca distribuível**.

12.1 Por que empacotar código em bibliotecas

O material extra sobre publicação de bibliotecas, de autoria de Rogério Pazetto Saldanha da Gama sob orientação da Profa. Amanda Lemette, organiza-se em quatro perguntas: por que usar biblioteca; como é o processo de importação; como publicar; e como usar a biblioteca publicada.

A motivação é direta. Uma função resolve o problema da repetição **dentro de um arquivo**. Um módulo resolve o problema **entre arquivos da mesma máquina**. Mas e quando o código deve estar disponível para **várias máquinas**? A solução apresentada é tornar o código acessível online.

12.2 O processo de publicação

Os slides descrevem dois passos:

1. **Gerar o arquivo `setup.py`**, que descreve o pacote (nome, versão, dependências, módulos incluídos);
2. **Publicar o código-fonte**, hospedando-o no GitHub.

A instalação da biblioteca publicada é então feita pelo gerenciador de pacotes, apontando para o repositório ou para o índice de pacotes.

```
pip install nome_da_biblioteca
```

Do ponto de vista conceitual, esta aula fecha o ciclo iniciado na introdução ao Python: funções são o primeiro passo para a reutilização de código, e a publicação de bibliotecas é o último, tornando esse reuso escalável para além do computador do autor.

13 Elaboração de Dashboards com Power BI

Esta é a aula mais extensa do bloco de Business Intelligence, conduzida pelo Prof. Anderson Nascimento. Ela reúne dois conjuntos de slides (Técnicas de Análise e Visualização de Dados I e II) e duas oficinas práticas.

13.1 Contextualização

O ponto de partida são os problemas enfrentados pelas organizações:

- Limitações quanto ao nível de detalhe e ao tempo para obtenção de informações;
- Reduzida capacidade de simulação e análise de dados;
- Custos e dependência crescentes para a formatação de relatórios e análises;
- Perda de desempenho;
- Dependência de terceiros para customização de relatórios.

A tendência de mercado responde a isso: espera-se que o usuário se concentre nos dados, e não na tecnologia, e a interface precisa ser intuitiva e fácil de usar. Daí o conceito de **BI Self-Service**, o autoatendimento em BI, em que os usuários têm o poder de criar relatórios e consultas quando e onde precisarem.

13.2 Onde ficam os painéis no processo de BI

Os painéis vêm **depois** do projeto de Data Warehouse. Todo o processo que envolve o planejamento da construção do DW e do Data Mart é essencial para garantir a **qualidade dos dados**; somente assim haverá aproveitamento adequado das técnicas de BI, seja na visualização, seja na mineração. Por isso a fase de **ETL** é tão importante. O material usa como referência o esquema de processo de BI adaptado de Krmac (2011), com sua *stage area*.

13.3 Consultas ad-hoc versus mineração de dados

É importante diferenciar os tipos de consulta realizados sobre um Data Warehouse ou Data Mart daqueles feitos por ferramentas de ciência de dados:

- As **consultas ad-hoc** são específicas para cada situação e respondem às perguntas dos analistas;
- As **consultas de mineração de dados** são responsáveis tanto pela pergunta quanto pela resposta, identificando correlações por meio das técnicas e tarefas empregadas.

No BI clássico, o que se tem é o BI Self-Service.

13.4 Cubos e dashboards

A informação pode ser disponibilizada por soluções corporativas, soluções departamentais, dashboards e cubos de visualização.

Cubos são ferramentas que permitem visualizar a informação em várias dimensões. Um cubo é composto de células, organizadas por grupos de medidas e dimensões. Uma célula representa a interseção lógica exclusiva, no cubo, de um membro de toda dimensão. O exemplo citado é um cubo com um grupo de medidas contendo duas medidas, organizadas junto a três dimensões: Origem, Rota e Temporal.

Dashboards são painéis que possibilitam o acompanhamento de indicadores e informações importantes para a tomada de decisão. São painéis de controle com informações consolidadas sobre os aspectos mais relevantes do negócio; constituem uma técnica de análise e visualização de dados; permitem responder rapidamente às principais questões da organização; podem ser customizados conforme as necessidades do negócio; e visam compartilhar informações afins com um grupo específico de usuários.

13.5 Dez dicas para um dashboard eficiente

1. Defina indicadores (KPIs) e exiba-os nos dashboards para os analistas do negócio;
2. Construa dashboards elegantes, mas objetivos;
3. Mantenha as informações online e atualizadas, para explorar todo o potencial da ferramenta;
4. Concentre as informações mais importantes em apenas uma aba ou tela;
5. Organize os objetos dentro de uma sequência lógica de leitura e interpretação dos dados;
6. Invista tempo no entendimento das informações necessárias para cada cliente; personalização é tudo;

7. Use ferramentas que possibilitem a extração de relatórios (Excel, PDF) com base nos filtros definidos pelo cliente;
8. Seja cauteloso quanto à quantidade de cores e efeitos;
9. Estude a relevância de cada item no dashboard;
10. Utilize a técnica da **leitura em Z**, que segue o percurso natural do olhar sobre a tela.

13.6 Ferramentas de mercado

As ferramentas de visualização facilitam a elaboração de consultas ad-hoc e permitem a visualização rápida e prática dos dados. Apesar de simples de operar, exigem grande esforço dos profissionais envolvidos até se chegar à camada de apresentação. Os principais players citados:

- **Power BI**: ferramenta da Microsoft, criada em 2015, para análise e compartilhamento de dados;
- **Tableau**: surgida na Califórnia em 2003, usa interface de arrastar e soltar e permite conectar a maioria dos bancos de dados e tabelas, criando quadros e visualizações interativas;
- **Oracle Data Visualization**: solução com mais de 20 tipos de objetos gráficos, incluindo gráficos, mapas e nuvens de tags, o que permite boa qualidade no processo de Data Discovery, também com recurso de arrastar e soltar;
- **Qlik Sense / QlikView**: criados pela QlikTech, fundada em Lund, na Suécia, em 1993. O QlikView é uma ferramenta de BI pioneira na plataforma associativa in-memory.

O **Quadrante Mágico do Gartner** é o relatório de pesquisa de mercado publicado pela consultoria americana Gartner, com o objetivo de fornecer uma análise qualitativa do mercado, sua direção, maturidade e participantes.

13.7 Power BI: produtos, limitações e fontes de dados

O Power BI é um pacote de ferramentas de análise e visualização de dados da Microsoft criado em 2015, que popularizou o termo BI Self-Service. Possui três produtos principais: **Power BI Desktop**, **Power BI Pro** e **Power BI Premium**.

A grande diferença entre a versão Desktop e a Pro é a possibilidade de **publicação e compartilhamento**. Na versão Pro em diante, é possível ter uma equipe com privilégios para realizar manutenção nos projetos, criar outros *workspaces* que permitam colaboração, além de criar aplicativos e gateways pessoais de dados.

As limitações apontadas: é preciso uma conta corporativa ou educacional para criar uma conta Power BI; não é obrigatório criar a conta, mas para publicar o dashboard é necessário tê-la; e há um trial de 60 dias para publicar o dashboard.

O Power BI dá suporte aos mais diversos arquivos de dados e SGBDs, entre eles XML, CSV, MDB, bancos SQL, JSON, PDF e Google Planilhas.

13.8 Funcionalidades principais

A interface do Power BI Desktop organiza-se em três visualizadores: **Visualizador de Dashboard**, **Visualizador de Dados** e **Visualizador de Relacionamentos**. Os slides destacam os pontos de acesso a bases de dados, a inserção manual de dados, o acesso ao **Power Query**, o painel de ferramentas visuais, os campos já carregados, os filtros e a formatação de objetos, além da área de análise de dados.

13.9 Gráficos

A elaboração de gráficos é extremamente simples: nos slides, um gráfico é gerado com três cliques a partir da planilha `pedidos.xlsx`. É possível mudar o tipo do gráfico com apenas um clique, bastando ter o gráfico selecionado e escolher o novo tipo, por exemplo passando de colunas para barras clusterizado.

Gráficos de pizza são populares e permitem visualizar o dado sob a perspectiva de participação percentual; há também os gráficos de rosca. Mas o material deixa um alerta memorável: **cuidado com gráficos com nome de comida**. A observação, aparentemente jocosa, reflete uma crítica consolidada em visualização de dados: gráficos de pizza e rosca dificultam a comparação precisa entre categorias.

Outros recursos apresentados:

- Incluir uma **linha mediana** para facilitar a identificação do valor médio de unidades, bem como linhas máximas e mínimas;
- Escolher a **ordem de exibição** dos dados, crescente ou decrescente, e a categoria de classificação;
- Filtrar o resultado com a opção **N-Superior**, exibindo apenas os maiores ou menores resultados, definindo antes qual campo será usado na filtragem;
- Optar por exibir o **valor total** de cada projeto em vez do percentual, já que o percentual de cada item difere conforme o recorte da análise.

A advertência central: é muito importante saber aplicar o gráfico certo para obter o insight certo.

13.10 Operações Drill-Down e Drill-Up

No Power BI é fácil trabalhar com hierarquias como a de tempo, o que permite visualizar as operações com mais ou menos detalhe. Chamamos de **Drill-Down** a operação de descer na hierarquia e de **Roll-Up** (ou Drill-Up) a de subir. A diferença é nítida: sem drill-down, os dados aparecem agregados por ano; com o drill-down habilitado, é possível explorar apenas um trimestre específico.

13.11 Tratando dados no Power BI

Por meio do **Editor de Consultas** do Power BI é possível realizar diversas transformações, para deixar a fonte de dados aderente ao propósito da análise. Os tratamentos exercitados sobre a base `pedidos.xlsx`:

- **Categorizar** o campo Estado, no menu Modelagem, fora do Power Query. A categorização informa ao Power BI que o campo representa uma localização geográfica, o que habilita mapas;
- Resolver o problema do nome do estado, transformando a sigla da UF no nome por extenso;
- Criar um campo novo com a sigla dos estados, utilizando o recurso de **duplicar coluna**;
- Tratar UFs que não estão na base.

13.12 Agrupamento de dados e hierarquias

É possível criar **agrupamentos** de dados manualmente no Power BI, o que é útil para viabilizar outros tipos de análise. Os exemplos trabalhados: criar um novo grupo para obter um campo **Região**, permitindo análises regionais; e criar um grupo chamado **Seção** para categorizar os produtos em Cozinha, Sala e Eletrônicos, com a tarefa de gerar em seguida um gráfico com o percentual do total de vendas dessas categorias.

O Power BI também permite criar **hierarquias** próprias. As tarefas propostas são hierarquizar Seção e Produtos, criando um gráfico de barras que permita operações de Drill Down e Drill Up, e hierarquizar Região e Estado, criando um gráfico com essas hierarquias.

13.13 Tabelas, matrizes e segmentação de dados

Tabelas e matrizes são objetos muito úteis para ver dados sob a perspectiva de detalhamento, mas é preciso cuidado ao encaixá-los em um dashboard, porque competem por espaço com as visualizações gráficas. Esses objetos podem ser facilmente exportados para o Excel em arquivo CSV.

A ferramenta **Segmentação de Dados** permite criar filtros, objetos fundamentais para a construção de dashboards, pois permitem ao usuário escolher os dados que deseja visualizar na tela. É importante perceber que a ativação de um filtro pode implicar a seleção de dados dos outros. Esses objetos podem ser exibidos como caixas de seleção, listas suspensas ou botões, e é possível definir se a lista terá as opções Selecionar Tudo e Seleção Única.

13.14 Linguagem DAX

A linguagem **DAX (Data Analysis Expressions)** é a linguagem usada pelo Power BI para a criação de fórmulas e funções. O DAX traz uma coleção de funções, operadores e constantes que podem ser usados em uma fórmula, ou expressão, para calcular e retornar um ou mais valores. Em resumo, o DAX ajuda a criar novas informações a partir de dados já presentes no modelo. É bastante parecida com as fórmulas do Microsoft Excel.

Há uma distinção importante de contexto de uso:

- A **M-Language** é usada quando se está no Query Editor, por exemplo para criar uma coluna personalizada;
- O **DAX** é usado no Data View, isto é, na exibição de dados.

13.15 Campos calculados com DAX

Em BI, eventualmente é preciso criar campos calculados, ou seja, novos campos baseados em campos carregados das fontes de dados. Na guia Exibição de Dados, clica-se em Dados e, na tabela Pedidos, em nova coluna; o Power BI aguarda então a edição da fórmula.

A base possui as colunas `PrecoUnidade` e `Unidades`, mas não uma coluna com o valor total de cada pedido. A fórmula:

```
Total = [PrecoUnidade] * [Unidades]
```

Com essa informação, já é possível calcular, usando o objeto **cartão**, o total histórico arrecadado em pedidos ao longo do tempo.

Um segundo exemplo introduz lógica condicional. Suponha que um pedido de 50 ou mais itens seja considerado "pedido alto" e abaixo disso, "normal". A fórmula usa a função condicional `IF`:

```
Tipo Pedido = IF([Unidades]>=50,"Alto","Normal")
```

Se for preciso uma terceira faixa (até 10 unidades: Normal; de 11 a 49: Moderado; acima de 50: Alto), passa-se ao **aninhamento condicional**, ou aninhamento de IFs:

```
Tipo Pedido = IF([Unidades]<=10,"Normal", IF([Unidades]>=50,"Alto","Moderado"))
```

O próprio Power BI auxilia na montagem da função, com sugestões durante a digitação.

13.16 Medidas

Medidas são indicadores que podem ser utilizados no dashboard. A diferença essencial em relação às colunas calculadas é que a medida **gera um único valor**, e não um valor para cada linha da base de dados. É possível criar medidas para total de vendas, média de vendas ou qualquer outro valor útil à análise. Uma boa prática recomendada é criar uma tabela específica para guardar as medidas, usando o botão Inserir Dados.

Dois exemplos:

```
Total de Pedidos = COUNTROWS(Pedidos)
Média = AVERAGE(Pedidos[Total])
```

Depois de criadas, as medidas também podem ser utilizadas em objetos como tabelas, matrizes e gráficos.

13.17 Tarefas e estudos de caso

Ao longo do material são propostas várias tarefas incrementais: criar um gráfico de linha que mostre o valor total de vendas por mês (o gráfico de linha deve ser usado para mostrar dados ao longo do tempo, sendo preciso muito cuidado ao escolher esse tipo de gráfico); criar uma visualização com os 5 estados com mais pedidos e os 5 com menos pedidos; usar a ferramenta **Treemap** para criar um gráfico que mostre o valor total de vendas por vendedor; e produzir um dashboard completo como estudo de caso.

13.18 Oficina 01

A partir da planilha `vendaCarros.xlsx`, deve-se criar um dashboard que exiba:

- **Gráficos**: percentual de vendas por estado; custo por fabricante; cores mais vendidas; vendas por ano;
- **Filtros**: fabricante, estado, modelo e cliente;
- **Cartões**: total de vendas por ano (coluna `ValorVenda`) e total de descontos aplicados (coluna `TotalDesconto`);
- **Dashboard**: elaborar apenas um painel e criar um template com cabeçalho e título.

13.19 Oficina 02

A partir da planilha `servicosdeti.xlsx`, deve-se criar um dashboard que exiba:

- **Gráficos**: serviços mais utilizados em percentual; evolução de atendimentos realizados por ano, usando hierarquia de data; cidades com mais clientes; atendimento por sexo;
- **Grupos**: criar um grupo chamado Área com a classificação Software (Configuração de Computador e Desenvolvimento de Software), Hardware (Manutenção de Computador, Manutenção de Impressora e Configuração de Rede) e Vendas (Venda de Equipamento e Venda de Acessório);
- **Campos calculados**: total do serviço, calculado como valor menos desconto;
- **Medidas**: valor médio de serviço, quantidade de serviços prestados e valor total de serviços prestados;
- **Filtros**: por área, por serviço, por data (usando slider), por cidade e por profissional.

As duas oficinas exercitam exatamente o encadeamento de conceitos da aula: carregar a base, tratar e agrupar dados, criar campos calculados e medidas com DAX, escolher visualizações adequadas e organizar tudo em um painel único com filtros interativos.

14 Operações com dataframes e criação de gráficos

Esta aula é dedicada à construção de gráficos com **Matplotlib** e ao trabalho com dataframes, com base no material da Profa. Amanda Lemette.

14.1 Matplotlib

O Matplotlib é uma biblioteca destinada à criação de gráficos estáticos, animados e interativos. A importação canônica e o exemplo-base da aula:

```
import matplotlib.pyplot as plt

years_x = [1975, 1980, 1985, 1990, 1995, 2000, 2005, 2010, 2015]
total_y = [1243, 1543, 1619, 1831, 1960, 2310, 2415, 2270, 1918]

plt.plot(years_x, total_y)
plt.show()
```

A lista `total_y` representa a quantidade de carbono emitido na atmosfera e `years_x` representa os anos. Os dados são ampliados com a produção de carvão e de gás natural:

```
coal_y = [823, 1136, 1367, 1547, 1660, 1927, 1983, 1827, 1352]
gas_y = [171, 200, 166, 175, 228, 280, 319, 399, 529]
```

14.2 Enriquecendo o plot

Os métodos apresentados são chamados **antes** de `plt.show()`:

- **Título**: `plt.title()`;
- **Títulos dos eixos**: `plt.xlabel()` e `plt.ylabel()`;
- **Marcações dos eixos**: `plt.xticks()` e `plt.yticks()`, além de `plt.xlim()` e `plt.ylim()` para definir os limites;
- **Legenda**: `plt.legend()`. É preciso primeiro adicionar rótulos a cada plot com o argumento `label`, e só então chamar a legenda;
- **Linhas de grade**: `plt.grid()`, como em `plt.grid(axis="y", linewidth=0.5)`.

14.3 Estilos e customização de linhas

O Matplotlib vem com muitos estilos prontos. É possível listá-los com `print(sorted(plt.style.available))` e aplicá-los com `plt.style.use()`. Dentro de um notebook, para isolar a configuração para um único plot sem generalizar para os demais, usa-se o gerenciador de contexto:

```
with plt.style.context('seaborn'):
    plt.plot(years_x, total_y)
    plt.show()
```

Sobre as linhas, há controle total dos objetos `Line2D`, com quatro propriedades principais: `color`, `marker` (padrão None), `linestyle` (padrão traço contínuo) e `linewidth` (padrão 1.5).

O exemplo integrador, com tamanho de figura definido:

```
plt.figure(figsize=(10,5))
plt.plot(years_x, total_y, label='total', c="grey", ls=':', marker='s')
plt.plot(years_x, coal_y, label='coal')
plt.plot(years_x, gas_y, label='gas')
```

```
plt.legend()
plt.title('CO2 emissions from electricity production - US')
plt.ylim((0,3000))
plt.ylabel('MtCO2/yr')
plt.grid(lw=0.5)
plt.show()
```

14.4 Axes versus Axis

Uma distinção conceitual que costuma confundir iniciantes:

- **Axis** é o eixo do plot, aquele que recebe as marcações e o título;
- **Axes** é a **área** dentro da qual o seu plot aparece.

É possível acessar a instância atual de Axes com `ax = plt.gca()`. Compare as duas formas de fazer a mesma coisa:

```
# interface pyplot
plt.plot(years_x, total_y)
plt.ylabel('MtCO2/yr')
plt.title('CO2 emissions from electricity production - US')
plt.show()

# interface via Axes
plt.plot(years_x, total_y)
ax = plt.gca()
ax.set_title('CO2 emissions from electricity production - US')
ax.set_ylabel('MtCO2/yr')
plt.show()
```

A diferença está nos nomes dos métodos: `matplotlib.pyplot.title` versus `matplotlib.axes.Axes.set_title`. Por que acessar `ax`? Por três motivos: para customização e refinamento; quando criamos múltiplos subplots; e para integração com outras bibliotecas, como o pandas.

14.5 Spines

As *spines* são as linhas de contorno do gráfico. É possível removê-las ou especificar uma cor:

```
ax.spines['right'].set_color(None)
```

Também é possível mudar a posição da linha de contorno:

```
ax.spines['bottom'].set_position(('axes', 0.5))   # metade do eixo y
ax.spines['bottom'].set_position(('data', 750))   # no valor 750 do eixo y
```

14.6 Figures, subplots e Axes

No Matplotlib, a **figure** significa toda a janela na interface do usuário. Dentro dessa figura pode haver vários subgráficos, organizados e numerados em uma grade de linhas e colunas. Os subplots pertencem à classe Axes; podem ser considerados equivalentes em primeira ordem. A única diferença é que se pode criar um `ax` sem uma grade de subgráficos, posicionando-o em posição absoluta dentro da figura.

Interface convencional:

```
plt.figure(figsize=(10,3))
plt.subplot(1,2,1)
plt.plot(years_x, coal_y, label="coal")
plt.plot(years_x, gas_y, label="gas")
plt.title('coal vs. gas')
plt.legend()
plt.subplot(1,2,2)
plt.plot(years_x, total_y, label="total", c='black')
plt.title("all energies")
plt.suptitle('US electricity CO2 emissions')
plt.show()
```

Interface orientada a objetos:

```
fig = plt.figure(figsize=(10,3))
ax1 = fig.add_subplot(1,2,1)
ax1.plot(years_x, coal_y, label="coal")
ax1.plot(years_x, gas_y, label="gas")
ax1.set_title('coal vs. gas')
ax1.legend()
ax2 = fig.add_subplot(1,2,2)
ax2.plot(years_x, total_y, c='black')
ax2.set_title('all energies')
fig.suptitle('US electricity CO2 emissions')
plt.show()
```

Nos documentos oficiais é comum encontrar o atalho por **atribuição por desestruturação**:

```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)

# ou, tratando os eixos como um array
fig, axs = plt.subplots(1, 2, figsize=(10,3))   # axs e um nd-array (1,2)
axs[0].plot(years_x, coal_y, label="coal")
axs[0].set_title('coal vs. gas')
axs[1].plot(years_x, total_y, c='black')
axs[1].set_title('all energies')
plt.suptitle('US electricity CO2 emissions')
plt.show()
```

14.7 Outros tipos de plots

Scatter plot. Gráfico de dispersão, que utiliza coordenadas cartesianas para exibir valores de um ou vários conjuntos de dados:

```
np.random.seed(19680801)   # fixando o estado aleatorio para reprodutibilidade
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```

Bar plot.

```
fig, ax = plt.subplots()
fruits = ['apple', 'blueberry', 'cherry', 'orange']
counts = [40, 100, 30, 55]
bar_colors = ['tab:red', 'tab:blue', 'tab:red', 'tab:orange']
ax.bar(fruits, counts, color=bar_colors)
ax.set_ylabel('fruit supply')
ax.set_title('Fruit supply by kind and color')
plt.show()
```

Histograma. Representação precisa da distribuição de dados numéricos, criada com `plt.hist()`:

```
import numpy as np
x = np.random.normal(size=10_000)
plt.hist(x, bins=100)   # o eixo horizontal traz as frequencias de cada bin
```

15 Tratamento de Dados no Power Query

Esta aula aprofunda a etapa de preparação de dados dentro do Power BI, retomando o material Técnicas de Análise e Visualização de Dados II com foco no **Editor de Consultas** (Power Query).

15.1 O papel do Power Query no fluxo

O Power Query ocupa, dentro do Power BI, o lugar que o ETL ocupa no processo clássico de BI: é onde os dados brutos são limpos, padronizados e enriquecidos antes de chegarem ao modelo e às visualizações. Através do Editor de Consultas, é possível realizar diversas transformações para deixar a fonte de dados aderente ao propósito da análise.

Um ponto de atenção conceitual destacado nos slides: **nem toda transformação acontece no Power Query**. A categorização do campo Estado, por exemplo, é feita no menu **Modelagem**, isto é, **fora** do Power Query. Já a criação de colunas personalizadas por meio da **M-Language** é feita dentro do Query Editor, ao passo que as fórmulas **DAX** operam no Data View. Compreender qual transformação pertence a qual camada evita retrabalho e modelos inconsistentes.

15.2 Transformações praticadas

Sobre a base `pedidos.xlsx`, as transformações exercitadas são:

- **Categorizar** o campo Estado no menu Modelagem, informando ao Power BI a natureza geográfica do campo;
- **Resolver o problema do nome do estado**, convertendo a sigla da UF no nome do estado por extenso;
- **Duplicar coluna** para criar um campo novo com a sigla dos estados, preservando o campo original;
- **Tratar UFs que não estão na base**, isto é, lidar com valores ausentes ou fora do domínio esperado.

Esse último ponto merece destaque: em qualquer projeto real de BI, a existência de valores fora do domínio (siglas inválidas, categorias não previstas, registros em branco) é a regra, não a exceção. A qualidade dos dados, tratada na primeira parte do módulo, é decidida exatamente aqui.

15.3 Do tratamento à modelagem

Uma vez tratados os dados, as etapas seguintes são as já descritas na aula de dashboards: **agrupamento de dados** (criação manual de grupos como Região ou Seção), **criação de hierarquias** (Seção com Produtos, Região com Estado) que habilitam Drill Down e Drill Up, e a criação de **campos calculados** e **medidas** com DAX. A sequência lógica completa é, portanto: carregar, tratar, categorizar, agrupar, hierarquizar, calcular e só então visualizar.

16 Manipulação de dados e noções de estatística com Python

Esta aula articula o material de gráficos com a manipulação de dataframes e uma revisão de estatística descritiva, apoiando-se nos mesmos slides de visualização de dados e nos notebooks de apoio da disciplina.

16.1 Do array ao dataframe

O percurso do curso em Python vai do escalar ao dataframe. As listas nativas introduzem a ideia de coleção; o NumPy introduz a computação vetorizada eficiente sobre arrays homogêneos; e o dataframe acrescenta a essa estrutura os **rótulos de linha e de coluna** e a possibilidade de colunas de tipos diferentes na mesma tabela, o que é exatamente o formato dos dados de negócio.

A integração com o Matplotlib é um dos motivos apontados nos slides para se trabalhar com o objeto `ax`: a interface orientada a objetos é o que permite que bibliotecas como o pandas desenhem seus gráficos dentro de eixos criados pelo usuário.

16.2 Noções de estatística sobre os dados

As funções de resumo já apresentadas ao longo do curso constituem a base da estatística descritiva aplicada:

- **Medidas de posição**: média, obtida com `mean()`; valores extremos, com `max()` e `min()`;

- **Contagens e proporções**: a Lei dos Grandes Números, exercitada na introdução ao Python, mostra empiricamente que a frequência relativa converge para a probabilidade teórica conforme a amostra cresce;
- **Distribuição**: o histograma, construído com `plt.hist()`, é a ferramenta gráfica correspondente, revelando a forma da distribuição dos dados;
- **Filtragem condicional**: `np.where()` permite classificar observações em categorias, como no exercício dos meses bons e ruins da demonstração financeira.

```
import numpy as np

lucro = receitas - custos
lucro_liquido = lucro * (1 - 0.30)
margem = lucro_liquido / receitas

media = lucro_liquido.mean()
meses_bons = np.where(lucro_liquido > media)
melhor_mes = lucro_liquido.max()
```

Esse bloco de código resume a filosofia da aula: **operações vetorizadas substituem laços**, e o resultado é código mais curto, mais legível e mais rápido. O exercício de desempenho com dois vetores de 100.000 elementos, medido com `time.process_time()`, dá a demonstração empírica dessa afirmação.

16.3 Combinando dados e gráficos

O passo final é usar as estatísticas calculadas para produzir visualizações informativas: um gráfico de linha para a evolução temporal do lucro, um gráfico de barras para comparar meses, um histograma para examinar a distribuição das margens. Vale a advertência que atravessa todo o módulo de visualização: é preciso saber aplicar o gráfico certo para obter o insight certo, e o gráfico de linha deve ser reservado para dados ao longo do tempo.

17 Análise exploratória de dados com Python

A aula final integra tudo o que foi visto, aplicando as bibliotecas **Matplotlib** e **Seaborn** à análise exploratória de bases reais.

17.1 Análises gráficas do repertório de EDA

O material lista os tipos de análise gráfica que compõem o repertório básico de uma análise exploratória:

- **Boxplot**;
- **Histogramas**;
- **Gráficos em barra**;
- **Gráficos de dispersão**;
- **Matriz de correlação**;
- entre outros.

A esses somam-se os gráficos de **densidade** e o **violinplot**, mencionados nos exercícios. Cada um responde a uma pergunta distinta: o histograma e a densidade descrevem a distribuição de uma variável; o boxplot resume essa distribuição em cinco números e evidencia outliers, além de permitir comparação entre grupos; o gráfico de barras compara categorias; o gráfico de dispersão examina a relação entre duas variáveis contínuas; e a matriz de correlação examina todas as relações lineares de uma vez.

17.2 Correlação

A **correlação** indica a força e a direção do relacionamento entre dois atributos. Trata-se de uma medida da relação entre dois atributos, mas o material faz questão de reforçar a advertência clássica: **correlação não implica causalidade**. Duas variáveis podem estar altamente correlacionadas sem que exista relação de causa e efeito entre elas. Na análise de correlação linear, o objetivo é determinar o **grau de relacionamento** entre duas variáveis, nada além disso.

Essa é uma das lições mais importantes de toda a disciplina para quem vai produzir sistemas de apoio à decisão: a ferramenta identifica associações, mas a interpretação causal exige conhecimento do domínio, exatamente como advertia o quarto equívoco comum apresentado na aula de Inteligência Artificial.

17.3 Estudo de caso: medicamento para ansiedade

O primeiro estudo de caso usa dados de um experimento sobre os efeitos de medicamentos para ansiedade e seu impacto em grupos que têm majoritariamente lembranças felizes ou tristes. As drogas e dosagens do experimento são:

- **A** — Alprazolam (Xanax, longo prazo), nas dosagens de 1 mg, 3 mg e 5 mg;
- **T** — Triazolam (Halcion, curto prazo), nas dosagens de 0,25 mg, 0,5 mg e 0,75 mg;
- **S** — Sugar Tablet (placebo), em 1, 2 ou 3 comprimidos.

A tarefa é carregar a base `Islander_data.csv` e fazer, livremente, análises gráficas para entendimento dos dados utilizando as bibliotecas matplotlib e seaborn. O material sugere ainda experimentar a biblioteca **pandas-profiling**, que gera automaticamente um relatório exploratório completo; para instalar no Colab, executa-se:

```
!pip install pandas-profiling==2.10.0
```

A presença de um grupo placebo no desenho experimental é o que permite atribuir causalidade aos efeitos observados. Sem ele, restariam apenas correlações.

17.4 Estudo de caso: tendências mundiais

O segundo exercício pede carregar a base `tendenciasMundiais.csv` em um dataframe e mostrar graficamente a relação entre **Expectativa de Vida** e **Taxa de Fertilidade** por país, para os anos de **1960** e **2013**, em um mesmo gráfico, utilizando a função `scatter`. Deve-se comparar e avaliar os dois anos, colocando legenda, título e nomes dos eixos. O aluno é convidado a criar outros gráficos para análise da base, usando boxplot, violinplot, histograma e demais recursos aprendidos.

```
import matplotlib.pyplot as plt

plt.scatter(fert_1960, vida_1960, label='1960', alpha=0.6)
plt.scatter(fert_2013, vida_2013, label='2013', alpha=0.6)
plt.xlabel('Taxa de Fertilidade')
plt.ylabel('Expectativa de Vida')
plt.title('Expectativa de Vida versus Fertilidade por pais')
plt.legend()
plt.show()
```

O exercício é elegante porque sobrepõe dois momentos históricos no mesmo espaço de coordenadas, tornando visível o deslocamento da nuvem de pontos ao longo de cinco décadas. Ele também exige aplicar, de uma só vez, praticamente todos os métodos de enriquecimento de plot vistos na aula de Matplotlib: rótulos de eixo, título, legenda e transparência para lidar com sobreposição.

18 Síntese da Disciplina

A disciplina Sistemas de Apoio à Decisão percorre, do início ao fim, a cadeia completa que transforma dado bruto em decisão informada. Ao final do curso, o aluno é capaz de **modelar e criar** um banco de dados relacional, **popular e manter** seus dados, **consultá-lo** com junções, agregações e subconsultas, **manipular e analisar** esses dados em Python com NumPy e dataframes, **visualizá-los** com Matplotlib e Seaborn, e **apresentá-los** em dashboards interativos construídos no Power BI.

Três fios conceituais atravessam todas as aulas. O primeiro é o da **estrutura**. Em SQL, ela aparece como chaves, cardinalidades e regras de mapeamento relacional, que garantem consistência e eliminam redundância. Em Python, aparece como a hierarquia escalar, vetor, matriz e tensor, e depois como o dataframe. No Power BI, aparece como o modelo de dados, com suas tabelas, relacionamentos e hierarquias. Em todos os casos, a lição é a mesma: dados bem estruturados tornam as perguntas fáceis de fazer, e dados mal estruturados tornam qualquer pergunta cara.

O segundo fio é o da **abstração e do reuso**. A disciplina parte de comandos isolados e sobe progressivamente: do comando SQL avulso à consulta parametrizada e à view; do trecho de script Python à função com parâmetros padrão, ao pacote e à biblioteca publicada no GitHub; do gráfico gerado com três cliques ao dashboard com filtros, medidas e hierarquias reaproveitáveis. Em cada camada, o esforço inicial de abstração é recuperado muitas vezes.

O terceiro fio é o da **interpretação responsável**. Ele aparece explicitamente em três momentos distantes entre si, e não por acaso: nos seis equívocos comuns da aula de Inteligência Artificial, que alertam contra a complexidade desnecessária e a falta de conhecimento do domínio; na advertência sobre gráficos de pizza e sobre a escolha do gráfico certo para o insight certo, no módulo de Power BI; e no aviso final de que **correlação não implica causalidade**, na aula de análise exploratória. A competência técnica de produzir números e gráficos é necessária, mas não suficiente. O que caracteriza um sistema de apoio à decisão bem construído é que ele apoie decisões corretas, e isso depende tanto do rigor na obtenção dos dados quanto da honestidade na sua leitura.

Por fim, vale observar a coerência entre os módulos aparentemente independentes. As funções de agregação do SQL (`SUM`, `COUNT`, `MAX`, `MIN`, `AVG`) reaparecem como métodos de NumPy (`sum()`, `mean()`, `max()`, `min()`) e como funções DAX (`COUNTROWS`, `AVERAGE`). O `GROUP BY` do SQL corresponde ao agrupamento de dados do Power BI e às operações de agregação sobre dataframes. O

`WHERE` corresponde ao `np.where()` e aos filtros e segmentações de dados. Não se trata de três linguagens diferentes, mas de três dialetos de um mesmo vocabulário analítico. Reconhecer essa correspondência é o que permite ao profissional migrar de ferramenta sem precisar reaprender os conceitos.