

Data Loading, Manipulation, and Visualization

Introdução Engenharia Informática

Mário Antunes

December 01, 2025

Universidade de Aveiro

Table of Contents i

Formatos e Estruturas de Dados

Carregamento e Manipulação de Dados

Visualização de Dados

Jupyter Notebooks

Leitura Adicional

Formatos e Estruturas de Dados

O Espectro da Estrutura de Dados i

Compreender a organização subjacente dos dados é o primeiro passo no processo (*pipeline*).

1. Dados Estruturados

- **Definição:** Dados que aderem a um modelo de dados pré-definido e, portanto, são diretos de analisar.
- **Formato:** Linhas e colunas (Tabular).
- **Esquema:** Rígido (*Schema-on-write*).
- **Exemplos:** Bases de Dados Relacionais (SQL), ficheiros Excel.

2. Dados Não Estruturados

- **Definição:** Informação que ou não tem um modelo de dados pré-definido ou não está organizada de uma maneira pré-definida.
- **Formato:** Texto, Binário, Media.
- **Esquema:** Nenhum (*Schema-on-read*).
- **Exemplos:** Documentos PDF, Vídeo, Áudio, e-mails em Texto Simples, publicações em Redes Sociais.

3. Dados Semi-Estruturados

- **Definição:** Dados que não residem numa base de dados relacional mas têm propriedades organizacionais que facilitam a sua análise. Utilizam “etiquetas” (*tags*) ou “marcadores” para separar elementos semânticos e impor hierarquias.
- **Formato:** Árvores hierárquicas ou pares Chave-Valor.
- **Exemplos:** JSON, XML, YAML, bases de dados NoSQL.

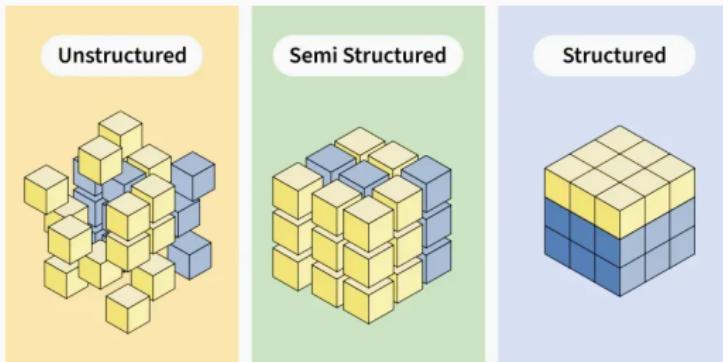


Figure 1: Ilustração da organização [Não | Semi]Estruturada

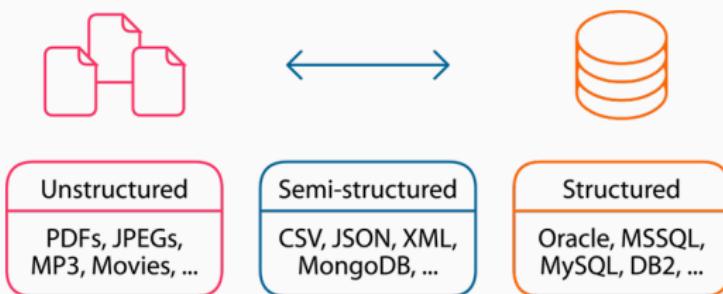


Figure 2: Exemplos de Ficheiros [Não | Semi]Estruturados

CSV (Comma Separated Values)

- **Estrutura:** Ficheiro de texto plano (*flat text*) onde as linhas são registos e as vírgulas separam as colunas.
- **Prós:** Universalmente suportado, extremamente leve.
- **Contras:** Sem suporte para tipos (tudo é uma *string*/número), sem aninhamento/hierarquia.

XML (eXtensible Markup Language)

- **Estrutura:** Estrutura baseada em árvore utilizando *tags* de abertura/fecho personalizadas.
- **Prós:** Padrão para serviços web legados (SOAP), suporta hierarquia complexa e esquemas (XSD).
- **Contras:** Verbooso (pegada de armazenamento pesada devido a *tags* repetidas), mais difícil de analisar (*parse*) do que JSON.

JSON (JavaScript Object Notation)

- **Estrutura:** Pares Chave-Valor utilizando chavetas {} para objetos e [] para arrays.
- **Prós:** O padrão para APIs Web modernas (REST), mapeamento nativo para Dicionários Python, legível por humanos.
- **Contras:** As chaves são repetidas para cada registo (verbooso).

YAML (YAML Ain't Markup Language)

- **Estrutura:** Baseia-se na indentação por espaços em branco para definir a hierarquia.
- **Prós:** O formato mais legível por humanos; perfeito para ficheiros de configuração (Docker, Kubernetes).
- **Contras:** Erros de indentação podem quebrar o ficheiro facilmente; a análise (*parsing*) pode ser mais lenta que JSON.

BSON (Binary JSON)

- **Estrutura:** Uma serialização codificada em binário de documentos tipo JSON.
- **Prós:** Otimizado para velocidade (traversia) e espaço; suporta tipos que o JSON não suporta (e.g., Date, BinData).
- **Contras:** Não legível por humanos sem um descodificador. Usado principalmente em MongoDB.

Exemplo: “Registo de Funcionário” i

Aqui está um conjunto de dados contendo dois registos representados em todos os formatos. Note como as “listas” de funcionários são tratadas.

1. CSV

Sem aninhamento nativo. A lista de “Competências” (*Skills*) requer um separador personalizado (e.g., pipe |).

```
id,name,skills,active
1,"Jane Doe","Python|SQL",true
2,"Bob Smith","Java|C++",false
```

Exemplo: “Registo de Funcionário” ii

2. XML

Requer uma *tag* raiz para envolver vários filhos.

```
<employees>
    <employee id="1">
        <name>Jane Doe</name>
        <skills><skill>Python</skill><skill>SQL</skill>
        </skills>
        <active>true</active>
    </employee>
    <employee id="2">
        <name>Bob Smith</name>
        <skills><skill>Java</skill><skill>C++</skill>
        </skills>
        <active>false</active>
    </employee>
</employees>
```

Exemplo: “Registo de Funcionário” iii

3. JSON

JSON usa parêntesis retos [] para denotar uma lista.

```
{"employees": [{  
    "id": 1, "name": "Jane Doe",  
    "skills": ["Python", "SQL"], "active": true  
},  
{  
    "id": 2, "name": "Bob Smith",  
    "skills": ["Java", "C++"], "active": false  
}]  
}
```

Exemplo: “Registo de Funcionário” iv

4. YAML

YAML usa traços – para denotar itens de uma lista.

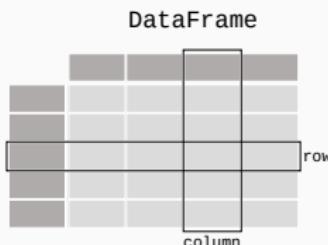
```
employees:
  - id: 1
    name: Jane Doe
    skills: [Python, SQL]
    active: true
  - id: 2
    name: Bob Smith
    skills:
      - Java
      - C++
    active: false
```

Carregamento e Manipulação de Dados

O Conceito de DataFrame

Um **DataFrame** é a estrutura de dados central em ciência de dados (usado por R, Pandas, Polars, Spark).

- **Modelo Conceptual:** Uma folha de cálculo em memória.
- **Estrutura:**
 - **Índice (Index):** Rótulos para linhas (eixo 0).
 - **Colunas (Columns):** Rótulos para variáveis (eixo 1).
 - **Células:** A interseção que contém os dados.
- **Homogeneidade:** Uma única coluna geralmente contém um único tipo de dados (e.g., todos Inteiros), mas colunas diferentes podem conter tipos diferentes.



- **Filosofia Central:** Execução *Eager* (Imediata). O código corre linha-a-linha imediatamente, tornando a depuração (*debugging*) e exploração intuitivas.
- **Arquitetura Chave:**
 - **Single-Threaded:** Corre principalmente num único núcleo de CPU.
 - **Baseado em Índices:** Depende fortemente de rótulos de linha explícitos (Índices) para alinhamento de dados, o que é crucial para análise de séries temporais.
- **Por que usar?**
 - **Ecossistema Inigualável:** É a entrada padrão para Scikit-Learn, Matplotlib e milhares de outras bibliotecas.
 - **Maturidade:** Se existe um problema de dados, existe uma resposta no StackOverflow sobre como resolvê-lo em Pandas.

- **Filosofia Central:** Avaliação Preguiçosa (*Lazy Evaluation*).
Constrói um plano de consulta otimizado antes da execução para minimizar trabalho e uso de memória.
- **Arquitetura Chave:**
 - **Multi-Threaded (Rust):** Escrito em Rust para contornar as limitações do Python, utilizando **todos os núcleos de CPU disponíveis** para processamento paralelo.
 - **Apache Arrow:** Usa um formato de memória colunar que permite transferência de dados *zero-copy* e *caching* eficiente.
- **Por que usar?**
 - **Desempenho:** Significativamente mais rápido que Pandas em grandes conjuntos de dados (acelerações de 10x-100x são comuns).
 - **Streaming:** Pode processar conjuntos de dados maiores que a RAM do seu computador.

Comparação de Bibliotecas: Pandas vs. Polars

Funcionalidade	Pandas	Polars
História	Criado em 2008. O padrão da indústria.	Mais recente (anos 2020). Construído para desempenho.
Backend Execução	Python/C / Cython. Eager : Corre linha-a-linha imediatamente.	Rust (Segurança e Velocidade). Lazy & Eager : Pode otimizar todo o plano de consulta antes de correr.
Paralelismo	<i>Single-threaded</i> (maioritariamente).	Multi-threaded (Paralelização nativa).
Memória	Copia dados frequentemente (Alto uso de RAM).	Formato de Memória Arrow (<i>Zero-copy</i> , eficiente).

Limpeza de Dados: Valores em Falta (Imputação) i

Os dados têm frequentemente lacunas (NaN ou Null). Deve decidir **removê-los** (*drop*) ou **preenchê-los** (*fill*). Os dados em falta podem ser preenchidos com a média (valores contínuos) ou moda (valores discretos).

- **Abordagem Pandas:**

```
# Drop rows with any missing values  
df.dropna()
```

```
# Fill with Mean (Imputation)  
mean_val = df['salary'].mean()  
df['salary'].fillna(mean_val, inplace=True)
```

Limpeza de Dados: Valores em Falta (Imputação) ii

- **Abordagem Polars:**

```
# Drop rows with any missing values  
df.drop_nans()  
  
# Fill with Mean  
df.with_columns(  
    pl.col("salary")  
        .fill_null(pl.col("salary")  
            .mean()))  
)
```

Limpeza de Dados: Outliers com IQR i

Outliers são valores extremos que se desviam de outras observações. Usamos o **Intervalo Interquartil (IQR)** para os detetar.

A Matemática:

1. **Q1 (25º Percentil):** O número médio entre o menor número e a mediana.
2. **Q3 (75º Percentil):** O número médio entre a mediana e o maior número.
3. **Fórmula IQR:**

$$IQR = Q_3 - Q_1$$

Limpeza de Dados: Outliers com IQR ii

O Filtro: O dado D é um *outlier* se:

$$D < (Q_1 - 1.5 \times IQR) \quad \text{ou} \quad D > (Q_3 + 1.5 \times IQR)$$

Implementação (Pandas):

```
Q1 = df['age'].quantile(0.25)
Q3 = df['age'].quantile(0.75)
IQR = Q3 - Q1
# Filtering
df_clean = df[~((df['age'] < (Q1 - 1.5 * IQR)) |
(df['age'] > (Q3 + 1.5 * IQR)))]
```

Implementação (Polars):

```
Q1 = df["age"].quantile(0.25, interpolation="linear")
Q3 = df["age"].quantile(0.75, interpolation="linear")
IQR = Q3 - Q1
# Filtering
df_clean = df.filter(
    pl.col("age").is_between(q1 - 1.5 * iqr, q3 + 1.5 * iqr)
)
```

Escalonamento de Dados (Scaling) i

O escalonamento dos dados pode ajudar a equilibrar o impacto de todas as variáveis no cálculo da distância e pode ajudar a melhorar o desempenho do algoritmo.



Figure 4: Scaling Data

Escalonamento de Dados (Scaling) ii

A. Escalonamento Min-Max (Normalização)

- **Objetivo:** Comprimir os dados num intervalo $[0, 1]$.
- **Fórmula:**

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

- **Uso:** Processamento de imagem (píxeis 0-255), Redes Neuronais.

B. Escalonamento Padrão (Padronização Z-Score)

- **Objetivo:** Centrar os dados em volta da Média=0 com Desvio Padrão=1.

Escalonamento de Dados (Scaling) iii

- **Fórmula:**

$$Z = \frac{X - \mu}{\sigma}$$

- **Uso:** PCA, Regressão Logística, Clustering.

Resumos (Tendência Central e Dispersão):

- **Média:** Média aritmética.
- **Mediana:** Valor central (Robusto a *outliers*).
- **Moda:** Valor mais frequente.
- **Quartis:** Pontos de verificação da distribuição (25%, 50%, 75%).

Correlação (Pearson):

- Mede a relação linear entre -1 (negativo perfeito) e $+1$ (positivo perfeito).
- 0 significa que não há correlação linear.

Exemplo de Código:

```
# Pandas/Polars Summary  
# Returns count, mean, std,  
# min, 25%, 50%, 75%, max  
print(df.describe())  
  
# Pandas Correlation Matrix  
print(df.corr())
```

Visualização de Dados

Matplotlib vs. Seaborn i

Matplotlib:

- **Papel:** O motor.
- **Filosofia:** "Tornar coisas fáceis, fáceis e coisas difíceis, possíveis."
- **Controlo:** Controlo granular sobre cada eixo, marca (*tick*), estilo de linha e elemento da tela (*canvas*).
- **Sintaxe:** Imperativa (Construção passo-a-passo).

Seaborn:

- **Papel:** A interface.
- **Filosofia:** "Desenhar gráficos estatísticos atraentes e informativos."

Matplotlib vs. Seaborn ii

- **Controlo:** Automatiza o mapeamento de cores, legendas e agregação estatística.
- **Sintaxe:** Declarativa (Descreva o *que* quer, não como desenhá-lo).

Melhores Gráficos para Tipos de Análise i

1. Análise: Distribuição (Univariada)

- **Objetivo:** Ver a dispersão e forma dos dados.
- **Gráfico:** **Histograma** ou **KDE** (Estimativa de Densidade por Kernel).
- **Exemplo:** `sns.histplot(data=df, x="price", kde=True)`

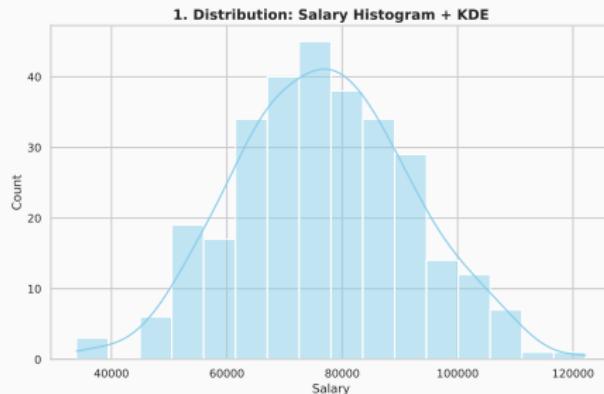
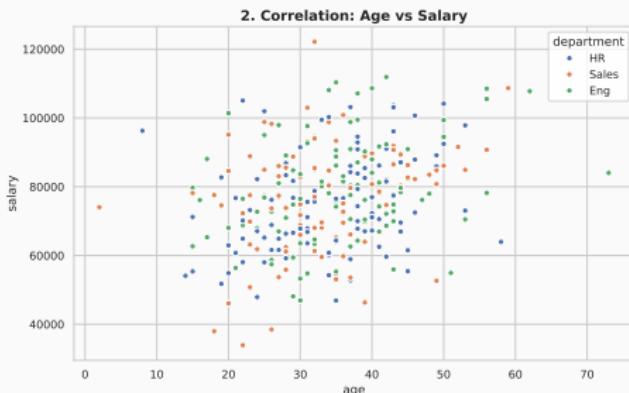


Figura 5: Distribuição

Melhores Gráficos para Tipos de Análise ii

2. Análise: Correlação (Bivariada)

- **Objetivo:** Ver como duas variáveis se relacionam.
- **Gráfico:** Gráfico de Dispersão (Scatter Plot) ou Mapa de Calor (Heatmap).
- **Exemplo:** `sns.scatterplot(data=df, x="age", y="salary")`



Melhores Gráficos para Tipos de Análise iii

3. Análise: Comparação (Categórica)

- **Objetivo:** Comparar valores entre grupos.
- **Gráfico:** Gráfico de Barras (agregados) ou Box Plot (distribuições).
- **Exemplo:** `sns.boxplot(data=df, x="department", y="salary")`



Melhores Gráficos para Tipos de Análise iv

4. Análise: Comparação (Densidade de Distribuição)

- **Objetivo:** Comparar a distribuição e probabilidade.
- **Gráfico:** Violin Plot.
- **Exemplo:** `sns.violinplot(data=df, x="department", y="salary")`

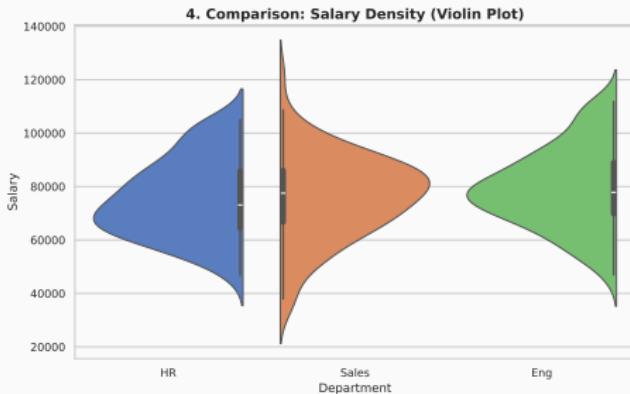


Figure 8: Comparison

Exportação: Raster vs. Vetor

Ao guardar as suas figuras (`plt.savefig`), a extensão determina a tecnologia.

Funcionalidade	Raster (Bitmap)	Vetor
Formatos	.png, .jpg, .bmp	.pdf, .svg, .eps
Composição	Grelha de píxeis coloridos.	Fórmulas matemáticas (caminhos, curvas).
Escalabilidade	Perde qualidade ao fazer <i>zoom</i> (pixeliza).	Escalabilidade infinita (nítido em qualquer <i>zoom</i>).
Tamanho de Ficheiro	Grande para alta resolução.	Pequeno (a menos que contenha milhares de pontos de dispersão).
Uso	Web, PowerPoints, Pré-visualizações rápidas.	Artigos Académicos, Impressão, Posters.

Jupyter Notebooks

A Estrutura Interativa i

Jupyter (JULia, PYThon, R) é uma plataforma de computação interativa baseada na web. É um documento JSON contendo uma lista ordenada de células de entrada/saída.

Funcionalidades Chave

- 1. Programação Literária:** Mistura código executável com texto narrativo (Markdown), equações (“*LaTeX*”) e imagens.
- 2. O Kernel:** O motor computacional (e.g., IPython) corre em segundo plano. Mantém o “estado” (variáveis permanecem em memória entre células).
- 3. Visualização:** Gráficos são renderizados em linha (*inline*) diretamente abaixo do código que os gerou.

Fluxo de Trabalho de Uso Eficaz i

1. A Célula “Markdown” Use para documentação. Suporta cabeçalhos (#), negrito (**), e listas para explicar a metodologia *antes* do código.

2. A Célula “Code” Corre Python. A última linha de uma célula é automaticamente impressa (não é necessário `print()`).

3. Comandos Mágicos

- `%matplotlib inline`: Incorpora gráficos no notebook.
- `%timeit`: Corre uma linha várias vezes para avaliar o desempenho (*benchmark*).

Fluxo de Trabalho de Uso Eficaz ii

- `!pip install X`: Corre comandos de *shell* para instalar bibliotecas dentro do notebook.

Leitura Adicional

Leitura Adicional

- **Documentação Pandas:** pandas.pydata.org - O guia definitivo.
- **Guia de Utilizador Polars:** pola.rs - Essencial para manipulação de dados de alto desempenho.
- **Galeria Seaborn:** seaborn.pydata.org - Inspiração visual para gráficos.
- **Matplotlib Anatomy of a Plot:** matplotlib.org - Compreender a hierarquia de objetos.
- **Projeto Jupyter:** jupyter.org