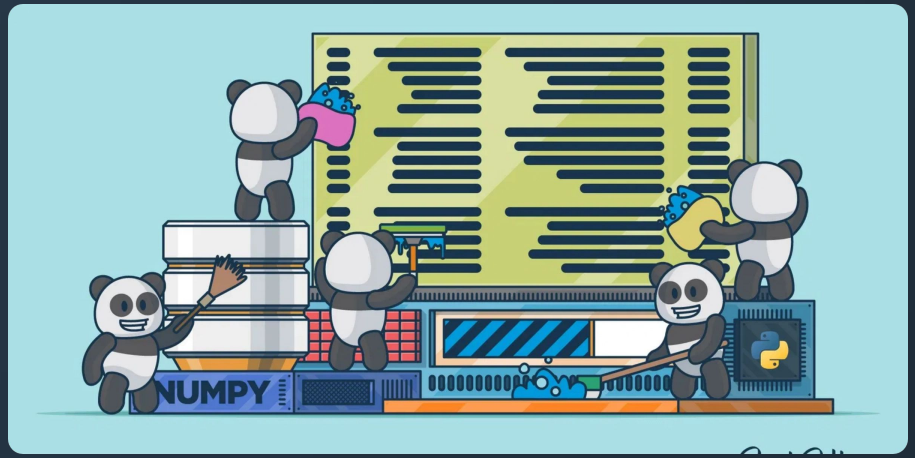
# Limpeza de Dados

## Duplicatas, Valores Ausentes e Inconsistências




Estratégias com Python, Pandas e NumPy

# O que é Limpeza de Dados?

**Definição:** Processo de identificar e corrigir erros, inconsistências e imprecisões em conjuntos de dados para garantir que sejam precisos, confiáveis e adequados para análise.

**Importância:** Dados limpos são a base para análises precisas, modelos de Machine Learning eficazes e tomadas de decisão informadas. Dados **sujos** podem levar a conclusões errôneas e resultados enganosos.

**Estatística:** Cientistas de dados gastam cerca de **60% a 80%** do seu tempo limpando e preparando dados antes de realizar análises.

# Tipos Comuns de Problemas de Dados

### Valores Ausentes (Missing Values)

Dados que não foram registrados ou estão em falta.

### Duplicatas (Duplicates)

Registros idênticos ou quase idênticos que aparecem mais de uma vez.

### Inconsistências (Inconsistencies)

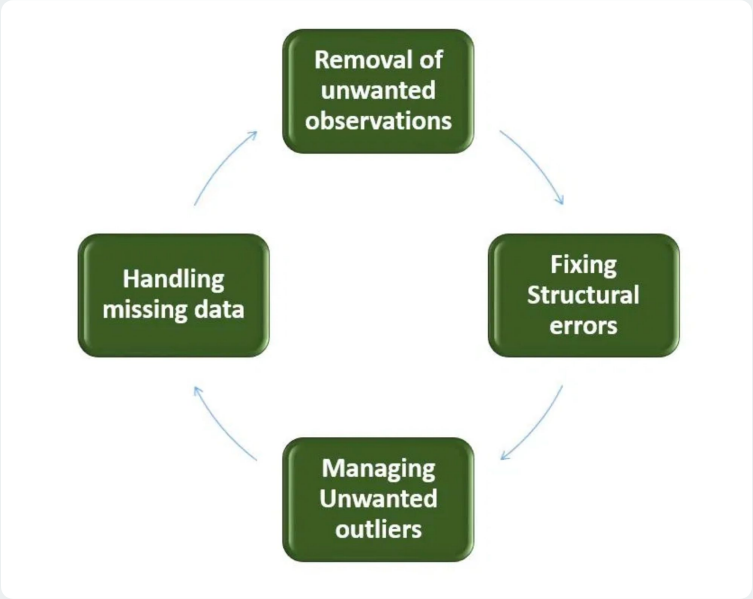
Variações na representação de dados (ex: 'USA' vs 'U.S.A.'), erros de digitação, formatos incorretos.

### Tipos de Dados Incorretos (Incorrect Data Types)

Colunas numéricas armazenadas como strings, datas como objetos, etc.

### Outliers

Valores extremos que se desviam significativamente do restante dos dados.



# Valores Ausentes: Identificação

**Problema:** Dados faltantes podem distorcer análises estatísticas e prejudicar o desempenho de modelos.

## Estratégia: Identificar a presença e a quantidade de valores ausentes por coluna

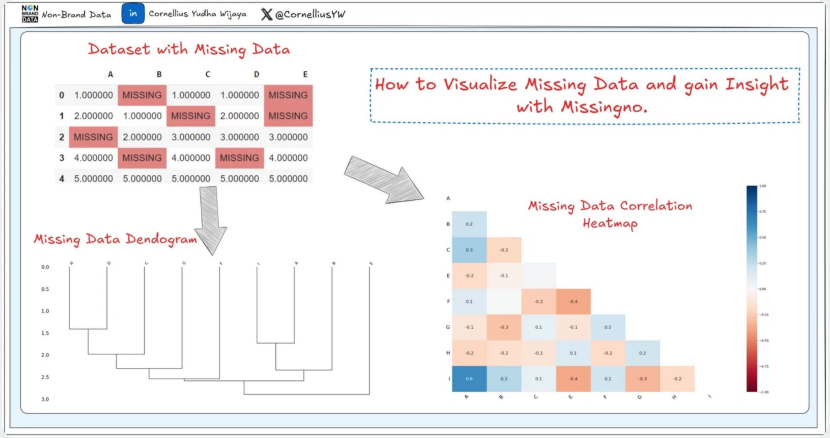
```python
import pandas as pd
import numpy as np

# Exemplo de DataFrame com valores ausentes
data = {'A': [1, 2, np.nan, 4, 5],
        'B': [np.nan, 2, 3, 4, np.nan],
        'C': [1, 2, 3, 4, 5]}
df = pd.DataFrame(data)

# Verificar valores ausentes por coluna
print("Valores ausentes por coluna:")
print(df.isnull().sum())

# Verificar porcentagem de valores ausentes
print("\nPorcentagem de valores ausentes:")
print(df.isnull().mean() * 100)
```

# Valores Ausentes: Tratamento (Remoção)

## Estratégia: Remover linhas ou colunas com valores ausentes

**Quando usar:** Se a quantidade de valores ausentes for pequena e não impactar significativamente o tamanho do dataset, ou se uma coluna tiver muitos valores ausentes e for irrelevante.

```
# Remover linhas com qualquer valor ausente
df_cleaned_rows = df.dropna()

# Remover linhas apenas se todas as colunas tiverem valores ausentes
df_cleaned_rows_all = df.dropna(how='all')

# Remover colunas com qualquer valor ausente
df_cleaned_cols = df.dropna(axis=1)

# Remover linhas com pelo menos 2 valores ausentes
df_cleaned_thresh = df.dropna(thresh=len(df.columns)-2)
```




Visualização de padrões de dados ausentes usando a biblioteca missingno

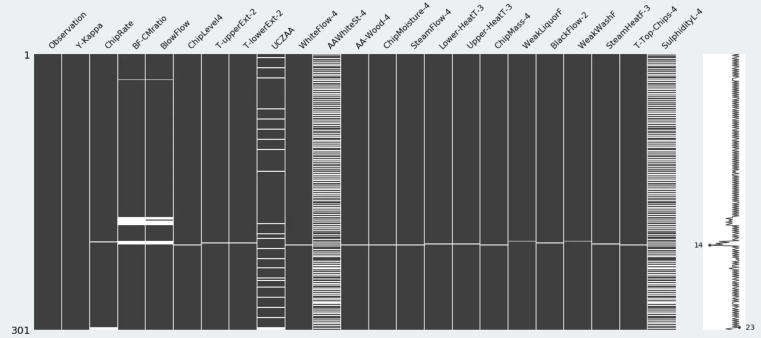
**Cuidado:** A remoção de dados pode introduzir viés se os valores ausentes não estiverem distribuídos aleatoriamente.

# Valores Ausentes: Tratamento (Imputação)

## Estratégia: Imputação de Valores

Quando a remoção resultaria em perda significativa de dados, podemos **preencher** os valores ausentes com estimativas.

```
# Preencher com a média da coluna
df_filled_mean = df.fillna(df.mean(numeric_only=True))

# Preencher com a mediana
df_filled_median = df.fillna(df.median(numeric_only=True))

# Preencher com um valor constante
df_filled_const = df.fillna(0)

# Interpolação (útil para séries temporais)
df_interpolated = df.interpolate()
```




*A escolha do método de imputação deve considerar a natureza dos dados e o impacto na análise posterior.*

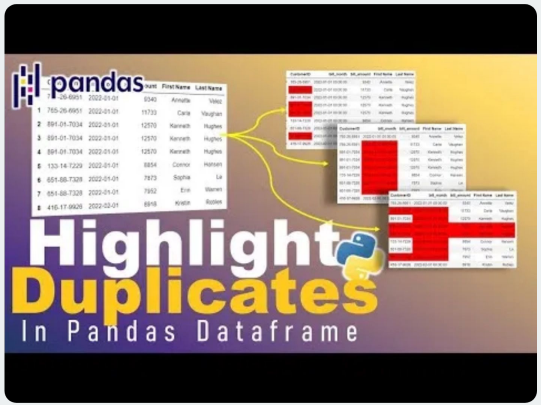
**Quando usar:** Quando a remoção resultaria em perda significativa de dados ou quando a ausência de dados não é aleatória.

# Duplicatas: Identificação

**Problema:** Registros duplicados podem superestimar a importância de certas observações e distorcer análises.

**Estratégia:** Identificar linhas que são cópias exatas ou duplicatas baseadas em um subconjunto de colunas.

```
# Exemplo de DataFrame com duplicatas
data_dup = {'ID': [1, 2, 2, 3, 4],
            'Nome': ['Alice', 'Bob', 'Bob', 'Charlie', 'David'],
            'Idade': [25, 30, 30, 35, 40]}
df_dup = pd.DataFrame(data_dup)


# Identificar linhas duplicadas (mantendo a primeira ocorrência como não duplicada)
duplicated_rows = df_dup[df_dup.duplicated()                                    ]

# Identificar duplicatas em um subconjunto de colunas
duplicated_on_subset = df_dup[df_dup.duplicated(subset=['Nome', 'Idade'])                                    ]
```




*O método duplicated() retorna uma série booleana onde True indica que a linha é uma duplicata de uma linha anterior.*

# Duplicatas: Tratamento (Remoção)

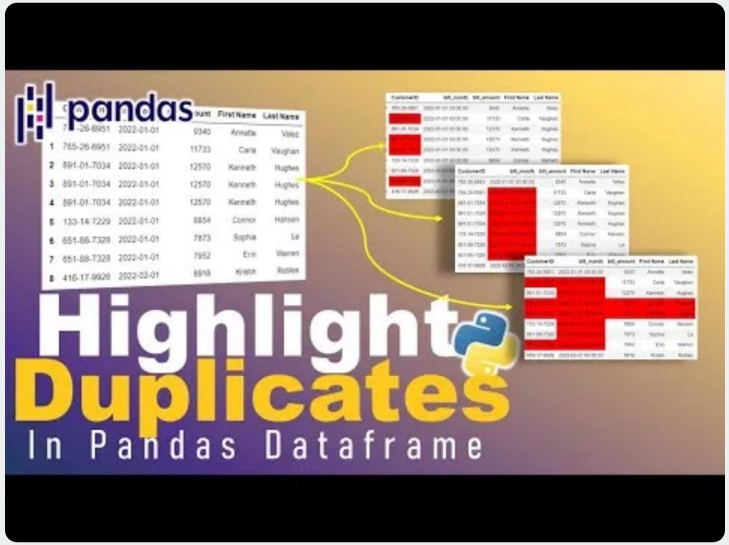
## Estratégia: Remover as linhas duplicadas do DataFrame

**Quando usar:** Quase sempre, a menos que as duplicatas representem observações válidas (ex: transações múltiplas).

```
# Remover todas as duplicatas, mantendo a primeira ocorrência
df_no_duplicates = df.drop_duplicates()

# Remover duplicatas, mantendo a última ocorrência
df_no_duplicates_last = df.drop_duplicates(keep='last')

# Remover duplicatas em um subconjunto de colunas
df_no_duplicates_subset = df.drop_duplicates(subset=['Nome', 'Idade'])
```

**Parâmetros importantes:**

- `keep` : 'first' (padrão), 'last' ou False (remove todas as ocorrências)
- `subset` : Lista de colunas para considerar na identificação de duplicatas
- `inplace` : Se True, modifica o DataFrame original




*Após a remoção de duplicatas, é importante verificar se o tamanho do DataFrame foi reduzido conforme esperado usando `len(df)` antes e depois da operação.*

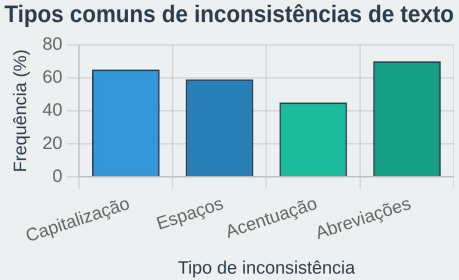
# Inconsistências: Padronização de Texto

**Problema:** Variações na capitalização, espaços extras, diferentes grafias para a mesma entidade.

## Estratégia: Padronizar texto usando métodos de string do Pandas

```
# Exemplo de DataFrame com inconsistências de texto
data_incons = {'Cidade': ['São Paulo ', 'são paulo', 'Rio de Janeiro', 'RIO DE JANEIRO ']}
df_incons = pd.DataFrame(data_incons)


# Converter para caixa alta
df_incons['Cidade_Upper'] = df_incons['Cidade'].str.upper()

# Converter para caixa baixa
df_incons['Cidade_Lower'] = df_incons['Cidade'].str.lower()

# Remover espaços em branco no início e fim
df_incons['Cidade_Strip'] = df_incons['Cidade'].str.strip()

# Padronizar para título (primeira letra maiúscula)
df_incons['Cidade_Title'] = df_incons['Cidade'].str.strip().str.title()
```

| Original | Upper | Lower | Strip | Title |
|---|---|---|---|---|

# Inconsistências: Mapeamento e Substituição

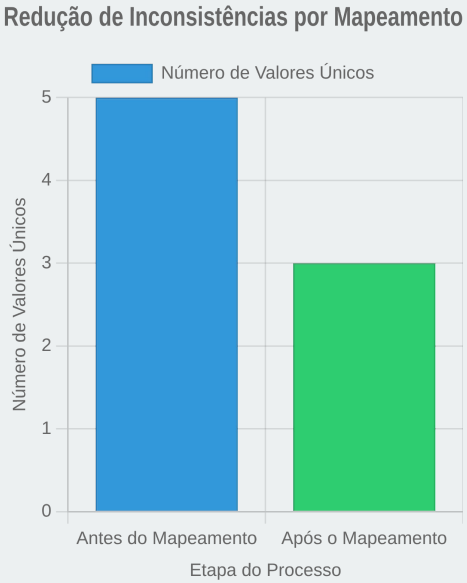
**Problema:** Múltiplas representações para o mesmo valor categórico (ex: 'EUA', 'USA', 'Estados Unidos') podem levar a análises incorretas e agrupamentos imprecisos.

## Estratégia: Criar um mapeamento de valores inconsistentes para um valor padrão

```
# Exemplo de DataFrame com inconsistências de mapeamento
data_map = {'País': ['USA', 'U.S.A.', 'Estados Unidos', 'Canadá', 'Mexico']}
df_map = pd.DataFrame(data_map)


# Mapear valores inconsistentes para um padrão
country_mapping = {'U.S.A.': 'USA', 'Estados Unidos': 'USA', 'Mexico': 'México'}

df_map['País_Padronizado'] = df_map['País'].replace(country_mapping)


# Alternativa usando o método map com dicionário e valor padrão
df_map['País_Padronizado2'] = df_map['País'].map(country_mapping).fillna(df_map['País'])
```

**Exemplo de Resultado:**

**Original:** ['USA', 'U.S.A.', 'Estados Unidos', 'Canadá', 'Mexico']
**Padronizado:** ['USA', 'USA', 'USA', 'Canadá', 'México']

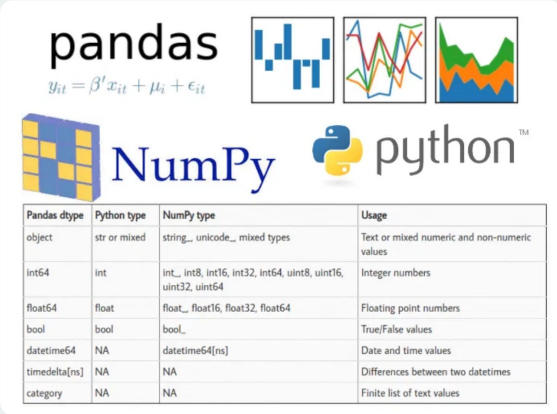


# Tipos de Dados Incorretos

**Problema:** Colunas com tipos de dados errados (ex: números como strings) impedem operações matemáticas ou causam erros.

## Estratégia: Converter colunas para o tipo de dado correto

```
# Exemplo de DataFrame com tipos de dados incorretos
data_types = {'Valor': ['10', '20', '30', 'quarenta'],
              'Data': ['2023-01-01', '2023-01-02', '2023-01-03', 'invalid-date']}
df_types = pd.DataFrame(data_types)

# Verificar tipos de dados atuais
print(df_types.dtypes)                    # Todos são 'object' (string)

# Converter 'Valor' para numérico, tratando erros
df_types['Valor_Numeric'] = pd.to_numeric(df_types['Valor'], errors='coerce')

# Converter 'Data' para datetime, tratando erros
df_types['Data_Datetime'] = pd.to_datetime(df_types['Data'], errors='coerce')
```




| Pandas dtype | Python type | NumPy type | Usage |
|---|---|---|---|
| object | str or mixed | string_, unicode_, mixed types | Text or mixed numeric and non-numeric values |
| int64 | int | int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64 | Integer numbers |
| float64 | float | float_, float16, float32, float64 | Floating point numbers |
| bool | bool | bool_ | True/False values |
| datetime64 | NA | datetime64[ns] | Date and time values |
| timedelta[ns] | NA | NA | Differences between two datetimes |
| category | NA | NA | Finite list of text values |

*Pandas oferece diversos tipos de dados otimizados para diferentes situações. Escolher o tipo correto melhora a performance e a precisão das análises.*

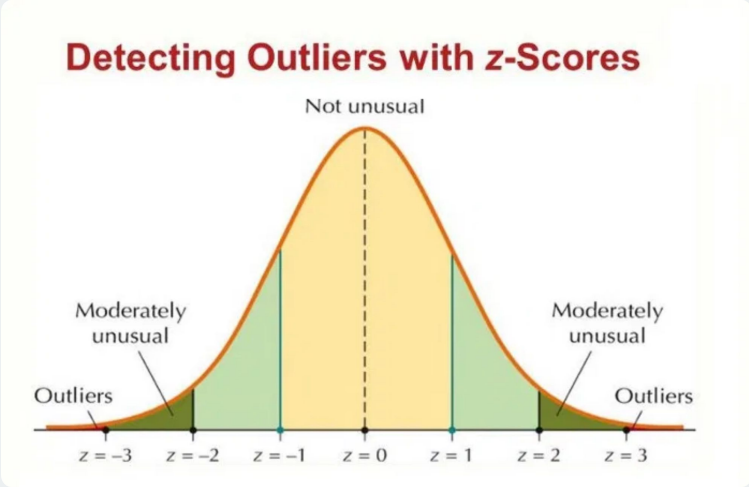
**Dica:** O parâmetro **errors='coerce'** converte valores problemáticos para NaN, permitindo identificar e tratar esses casos posteriormente.

# Outliers: Detecção e Tratamento

**Problema:** Outliers são valores extremos que se desviam significativamente do restante dos dados. Podem distorcer médias, variâncias e impactar negativamente modelos estatísticos e de ML.

## Métodos de Detecção

```
# Método Z-score
from scipy.stats import zscore
df['Z_score'] = zscore(df['Pontuação'])
outliers_zscore = df[df['Z_score'].abs() > 3]

# Método IQR (Interquartile Range)
Q1 = df['Pontuação'].quantile(0.25)
Q3 = df['Pontuação'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers_iqr = df[(df['Pontuação'] < lower_bound) |
(df['Pontuação'] > upper_bound)]
```




*A escolha do método de detecção e tratamento deve considerar o contexto dos dados e o objetivo da análise. Nem todos os outliers são erros - alguns podem representar informações valiosas.*

## Tratamento de Outliers

- **Remoção:** Eliminar outliers se forem erros ou não representativos

# Boas Práticas em Limpeza de Dados

### Entender os Dados

Sempre comece com uma exploração profunda dos dados (EDA) para compreender sua estrutura e características.

### Documentar

Registre todas as etapas de limpeza e as decisões tomadas para garantir reprodutibilidade e transparência.

### Manter o Original

Nunca modifique o dataset original diretamente; trabalhe em cópias para preservar os dados brutos.

### Automatizar

Use scripts para limpeza repetível e consistente, facilitando a aplicação do mesmo processo em novos dados.

### Validar

Após a limpeza, verifique se os dados estão consistentes e se os problemas foram resolvidos adequadamente.

### Iterar

A limpeza de dados é um processo iterativo, não linear. Esteja preparado para revisitar etapas anteriores.

# Atividade Prática

## Objetivo: Aplicar as técnicas de limpeza de dados em um dataset real

Nesta atividade, você irá trabalhar com o dataset **Titanic** do Kaggle, que contém informações sobre os passageiros do navio, incluindo se sobreviveram ou não ao naufrágio.

1. **Carregar o dataset** : Baixe o dataset do Titanic e carregue-o usando Pandas.

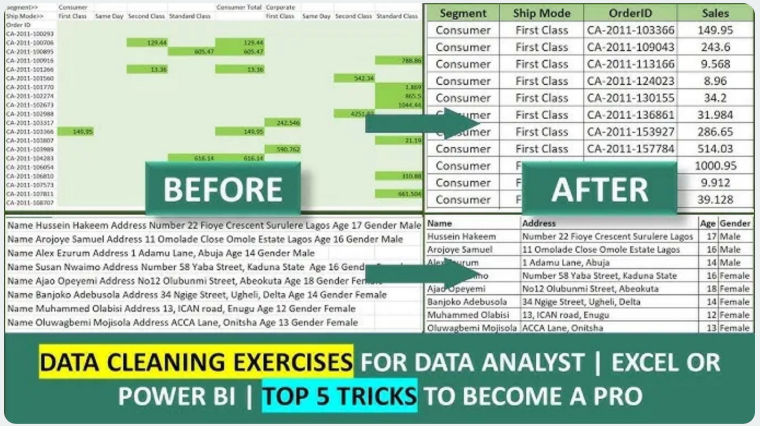
```
import pandas as pd
df = pd.read_csv('titanic.csv')
```

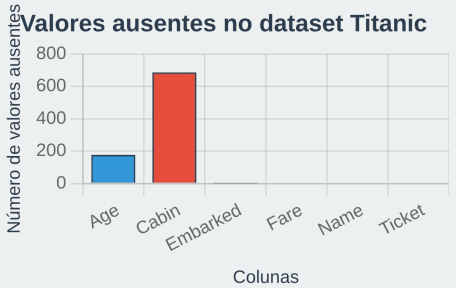
2. **Identificar problemas** : Analise o dataset para identificar valores ausentes, duplicatas e inconsistências.

3. **Tratar valores ausentes** : Aplique técnicas de imputação ou remoção para lidar com valores ausentes nas colunas 'Age', 'Cabin' e 'Embarked'.

4. **Padronizar dados** : Padronize os valores da coluna 'Sex' e 'Embarked', e converta tipos de dados conforme necessário.

5. **Documentar e justificar** : Documente todas as etapas realizadas e justifique suas escolhas de tratamento.




*Dados baseados no dataset Titanic do Kaggle, que contém 891 registros de passageiros.*

# Referências e Recursos Adicionais

## Livros


**McKinney, W. (2017).** *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython.* O'Reilly Media.

**Géron, A. (2019).** *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media.

**Chen, D. Y. (2022).** *Pandas for Everyone: Python Data Analysis.* Addison-Wesley Professional.


## Artigos e Tutoriais Online

*Pythonic Data Cleaning With pandas and NumPy*

https://realpython.com/python-data-cleaning-numpy-pandas/

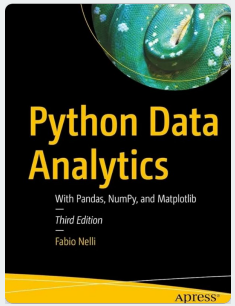
*A Beginner's Guide to Data Cleaning in Python*

https://www.datacamp.com/tutorial/guide-to-data-cleaning-in-python

*Practical Examples of Data Cleaning Using Pandas and Numpy*

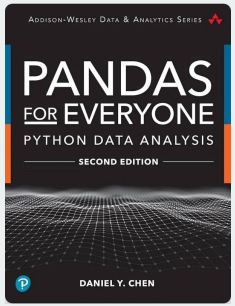
https://medium.com/pythoneers/practical-examples-of-data-cleaning-using-pandas-and-numpy-5f59021f0144

## Documentação Oficial



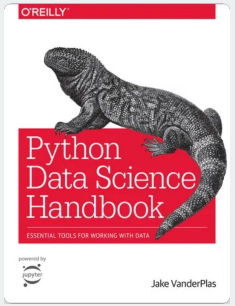

**Python Data Analytics**

Fabio Nelli




**Pandas for Everyone**

Daniel Y. Chen




**Python Data Science Handbook**

Jake VanderPlas

*Estes recursos fornecem informações detalhadas sobre técnicas de limpeza de dados e uso das bibliotecas Python para análise de dados.*