Limpeza e Manipulação de Dados Textuais (Strings) com Pandas

Transformando dados textuais caóticos em informações valiosas

Aula 06 - UC2

Ciência de Dados

Objetivos da Aula

- Compreender os desafios específicos dos dados textuais na ciência de dados e por que eles são considerados "a maior dor de cabeça" na área
- Aprender a utilizar as funções de manipulação de strings do Pandas através do acessor especial **.str** para limpeza e padronização de dados
- Desenvolver habilidades para detectar e corrigir erros comuns em dados textuais, como inconsistências de caixa, espaços invisíveis e erros de categoria
- Aplicar técnicas de padronização e transformação de strings em datasets reais, incluindo filtragem, extração e documentação de rotinas

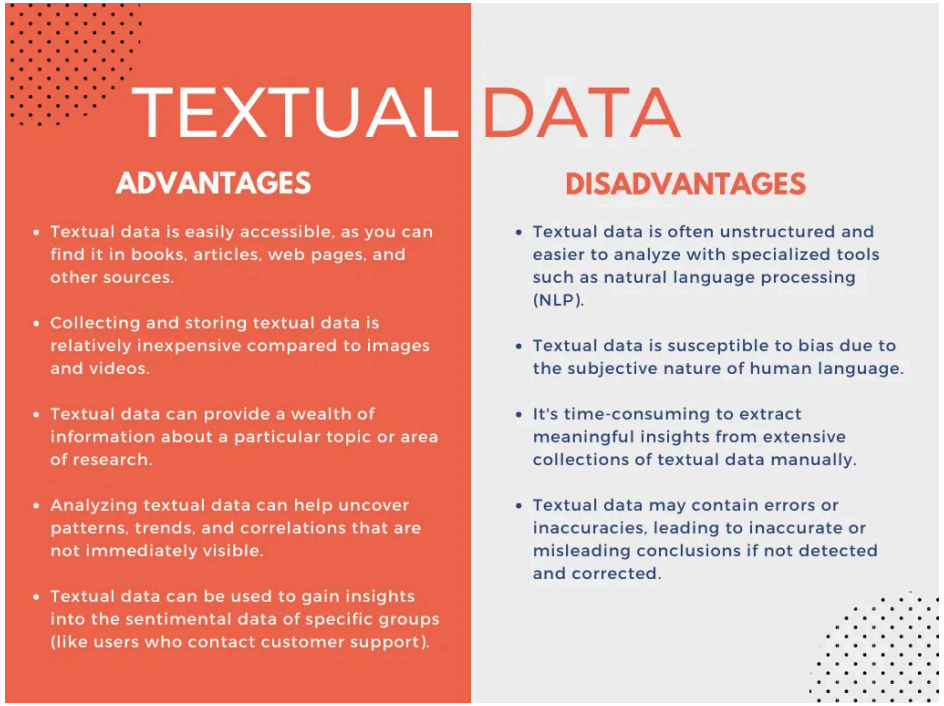
Ao final desta aula, você terá as habilidades necessárias para transformar dados textuais caóticos em informações estruturadas e confiáveis.

Fundamentos e Desafios dos Dados de Texto

"Por que dados textuais são a maior dor de cabeça na Ciência de Dados?"

Diferente dos dados numéricos, o texto tem **variações infinitas**. Erros de digitação, abreviações e diferentes formas de escrever a mesma coisa tornam a análise textual um verdadeiro desafio.

- ⚠ **Falta de padronização:** "São Paulo", "Sao Paulo", "SP", "SAO PAULO" - todos representam a mesma entidade, mas são tratados como valores diferentes pelo computador.
- ⚠ **Erros humanos:** Erros de digitação, abreviações inconsistentes e formatos variados são comuns em dados inseridos manualmente.
- ⚠ **Complexidade linguística:** Sinônimos, gírias, expressões regionais e contextos específicos tornam a interpretação automática difícil.
- ⚠ **Impacto nas análises:** Dados textuais não padronizados podem levar a contagens incorretas, agrupamentos falhos e conclusões equivocadas.



Vantagens e desafios dos dados textuais na análise de dados

O Universo das Strings no Pandas

Funções Essenciais para Manipulação de Strings

O Pandas oferece um conjunto poderoso de ferramentas para manipulação de strings através do **acessor .str**, que permite aplicar funções de manipulação de texto a todas as células de uma coluna de uma só vez.

</> df['coluna_texto'].str.método()

.lower() / .upper()

Padroniza o texto para minúsculas ou maiúsculas

```
df['Cidade'].str.lower() # são paulo, rio de janeiro
```

.strip() / .lstrip() / .rstrip()

Remove espaços em branco (ambos os lados, esquerda ou direita)

```
df['Cidade'].str.strip() # Remove " Rio " → "Rio"
```

.replace()

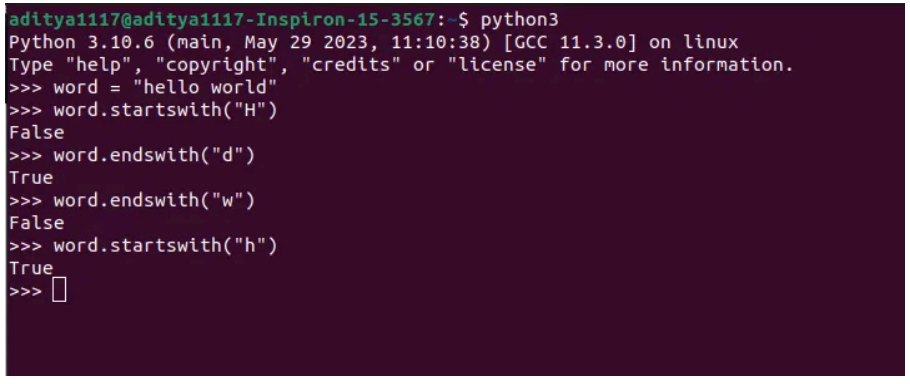
Substitui um texto por outro

```
df['Categoria'].str.replace('Fem', 'Feminino')
```

.contains()

Verifica se a string contém um determinado texto

```
df[df['Endereço'].str.contains('Avenida')]
```



Exemplos de manipulação de strings em Python

```
# Exemplo completo: Corrigindo uma coluna de cidades
# DataFrame original
df = pd.DataFrame({
    'Cidade': [' RIO ', 'sao paulo', ' Rio', 'SÃO PAULO']
})

# Aplicando correções
df['Cidade'] = df['Cidade'].str.lower().str.strip()
```

Atividade Prática 1: Detectar e Corrigir Erros Textuais

Objetivo da Atividade

Formato: Trabalho em grupos usando Google Colab
Dataset: Cadastro de clientes com erros propositais

1 Identificação de Inconsistências

Use **value_counts()** para identificar inconsistências:

```
df['Cidade'].value_counts()
```

2 Padronizar Caixa e Remover Espaços

Aplique **.str.lower()** e **.str.strip()**:

```
df['Cidade'] = df['Cidade'].str.lower().str.strip()
```

3 Corrigir Categorias com .replace()

Use **.str.replace()** para padronizar:

```
df['Cidade'] = df['Cidade'].str.replace('sp', 'são paulo')
```

4 Demonstrar o Antes/Depois

Compare os resultados usando **value_counts()**:

```
print("Contagem após limpeza:")
df['Cidade'].value_counts()
```

	song_title	lyrics
0	Easy Easy	[verse 1]\nwell same old bobby, same old beat\...
1	Baby Blue	[verse 1]\nmy sandpaper sigh engraves a line\n...
2	Out Getting Ribs	[verse 1]\nand hate runs through my blood\nwel...
3	Dum Surfer	[verse 1]\ndumb surfer is giving me his cash\n...
4	Border Line	[verse 1]\nyou know i tried so hard\nmy feelin...
...	...	...
79	What Stars?	please complete me\nit must be the answer\neve...
80	Comet Face (Live)	[intro]\ni don't really have too much to say\n...
81	Cellular (Live)	[verse 1]\nthere's a television\nthere's a tel...
82	Half Man Half Shark (Live)	[intro]\nhalf man with the body of a shark\nha...
83	The Wake	i know there must be something caught up in th...

Exemplo de limpeza e padronização de dados textuais

Dica para o Instrutor:

Prepare um dataset com erros comuns encontrados em dados reais, incluindo variações de nomes de cidades e categorias com inconsistências típicas.

Funções Avançadas para Manipulação de Strings

.str.split()

Divide uma string em uma lista de substrings com base em um separador.

```
df['Endereço'].str.split(',', expand=True)
```

.str.extract()

Extrai padrões específicos usando expressões regulares.

```
df['Email'].str.extract(r'(\w+)@(\w+\.\w+)')
```

.str.startswith() e .str.endswith()

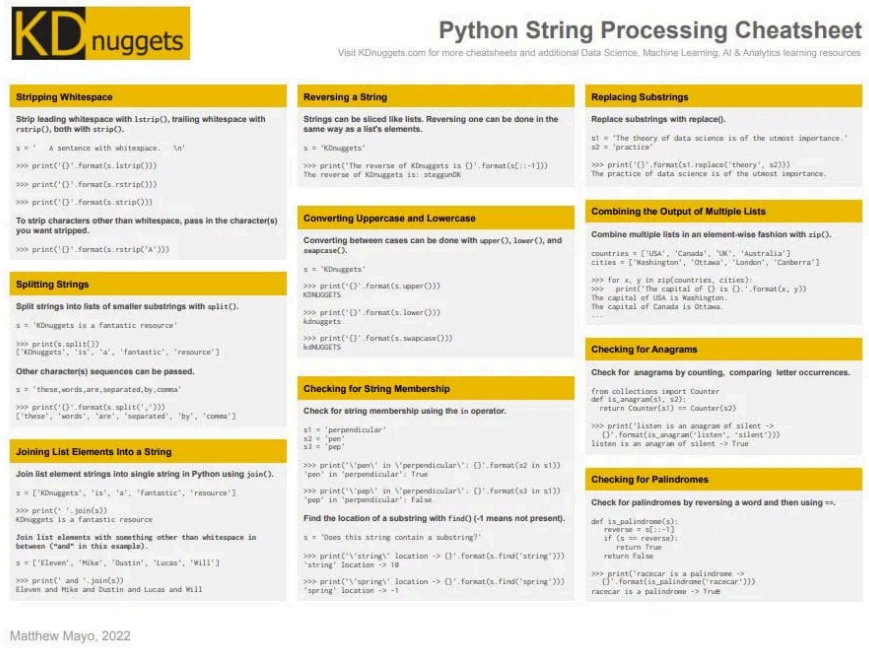
Verifica se as strings começam ou terminam com um padrão específico.

```
df[df['URL'].str.endswith('.gov')]
```

.str.replace() com regex

Substitui padrões complexos usando expressões regulares.

```
df['Telefone'].str.replace(r'[()-.\s]', '')
```



Referência de métodos para processamento de strings em Python

Atividade Prática 2: Filtragem e Extração de Informações

Instruções

Nesta atividade, vamos trabalhar com o dataset já limpo para extrair informações específicas.
Formato: Grupos usando Google Colab

1 Filtragem por Texto com .str.contains()

Filtre o DataFrame para manter apenas linhas com texto específico (ex: endereços com "Avenida").

2 Criação de Novas Colunas por Concatenação

Combine colunas textuais para criar uma nova (ex: Nome + Sobrenome = Nome_Completo).

3 Extração de Informações com .str.extract()

Use expressões regulares para extrair padrões específicos como códigos ou datas.

4 Justificativa e Apresentação

Explique a utilidade da transformação e apresente sua solução mais interessante.

</> Exemplo 1: Filtragem com .str.contains()

```
# Filtrar clientes que moram em Avenidas
clientes_avenidas = df[df['Endereco'].str.contains('Avenida',
case=False)]
```

</> Exemplo 2: Criação de Novas Colunas

```
# Criar coluna de nome completo
df['Nome_Completo'] = df['Nome'] + ' ' + df['Sobrenome']
```

</> Exemplo 3: Extração com Expressões Regulares

```
# Extrair código de área de telefones
df['Codigo_Area'] = df['Telefone'].str.extract(r'\((\d{2})\)')
```

```
Telefone: (11)98765-4321 → Codigo_Area: 11
Telefone: (21)99999-8888 → Codigo_Area: 21
```

Atividade Prática 3: Padronização de Formatos de Dados

Instruções

Nesta atividade, vamos trabalhar com a padronização de formatos de dados textuais como datas e números de telefone.
Formato: Grupos usando Google Colab

Datas: Conversão de Tipos

Use **pd.to_datetime()** para converter texto para formato de data:

```
# Converter coluna de datas em formato de texto
df['Data_Texto'] = ['01/05/2023', '02/05/2023', '03/05/2023']

# Converter para datetime
df['Data'] = pd.to_datetime(df['Data_Texto'], format='%d/%m/%Y')

# Agora podemos fazer operações de data
df['Mes'] = df['Data'].dt.month
```

Desafio do Grupo

1. Padronizar uma coluna de CEPs (ex: "01234-567", "01234567")
2. Converter datas em diferentes formatos para o padrão datetime
3. Criar função que padronize telefones e adicione código do país
4. Discutir: Por que a padronização é crucial para análises?

Telefones/CEP: Remoção de Caracteres

Use **.str.replace()** com expressões regulares para remover caracteres indesejados:

```
# Padronizar formato de telefones
df['Telefone_Original'] = ['(11) 98765-4321', '(21)99999-8888']

# Remover caracteres especiais
df['Telefone_Limpo'] = df['Telefone_Original'].str.replace(r'[()-.\s]',
'', regex=True)
```

Telefone_Original	Telefone_Limpo
(11) 98765-4321	11987654321
(21)99999-8888	21999998888

Impacto dos erros de formato: Se o ano estiver em formato de texto, o Pandas não consegue calcular a diferença entre datas, ordenar cronologicamente ou extrair componentes como mês e dia da semana.

Dica: Para formatos de data variados, use o parâmetro `errors='coerce'` no `pd.to_datetime()` para converter valores problemáticos em NaT (Not a Time) e depois identificar quais registros precisam de atenção especial.

Atividade Prática 4: Documentação de Rotinas e Relato dos Resultados

Boas Práticas e Dicas

Recomendações para Lidar com Dados Textuais

Sempre Verifique Valores Únicos Antes

Antes de iniciar qualquer limpeza, use `value_counts()`, `unique()` ou `nunique()` para entender a distribuição dos seus dados textuais.

```
df['Cidade'].value_counts()
df['Categoria'].unique()
df['Estado'].nunique() # Número de valores únicos
```

Crie Cópias de Segurança

Sempre mantenha uma cópia das colunas originais antes de aplicar transformações irreversíveis.

```
df['Cidade_Original'] = df['Cidade'].copy()
df['Cidade'] = df['Cidade'].str.lower().str.strip()
```

Documente Todas as Transformações

Mantenha um registro claro de todas as transformações aplicadas para garantir reprodutibilidade e facilitar a manutenção.

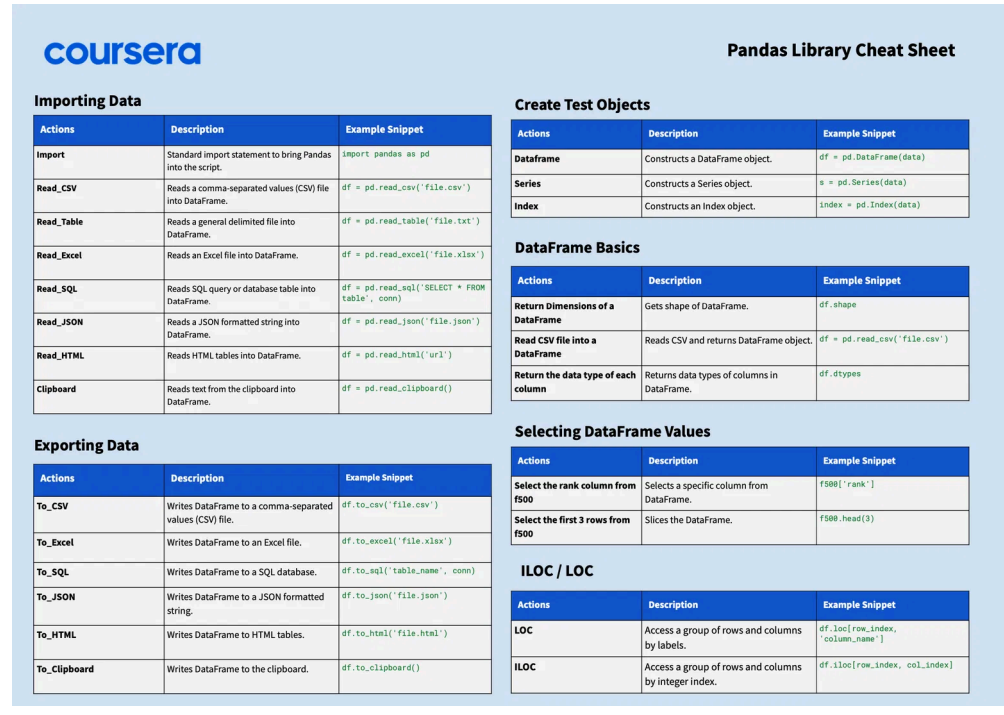
```
# Limpeza de dados da coluna Cidade
# 1. Converter para minúsculas
# 2. Remover espaços em branco
# 3. Padronizar nomes de cidades
```

Padronize Primeiro, Depois Filtre ou Agrupe

Sempre limpe e padronize seus dados textuais antes de realizar operações de filtragem, agrupamento ou análise.

Funções Úteis para Verificação

- `df['coluna'].unique()` - Lista todos os valores únicos
- `df['coluna'].nunique()` - Conta o número de valores únicos
- `df['coluna'].value_counts()` - Conta a frequência de cada valor
- `df['coluna'].isnull().sum()` - Conta valores ausentes



coursera

Pandas Library Cheat Sheet

Importing Data

Actions	Description	Example Snippet
Import	Standard import statement to bring Pandas into the script.	`import pandas as pd`
Read_CSV	Reads a comma-separated values (CSV) file into DataFrame.	`df = pd.read_csv('file.csv')`
Read_Table	Reads a general delimited file into DataFrame.	`df = pd.read_table('file.txt')`
Read_Excel	Reads an Excel file into DataFrame.	`df = pd.read_excel('file.xlsx')`
Read_SQL	Reads SQL query or database table into DataFrame.	`df = pd.read_sql('SELECT * FROM table', conn)`
Read_JSON	Reads a JSON formatted string into DataFrame.	`df = pd.read_json('file.json')`
Read_HTML	Reads HTML tables into DataFrame.	`df = pd.read_html('url')`
Clipboard	Reads text from the clipboard into DataFrame.	`df = pd.read_clipboard()`

Exporting Data

Actions	Description	Example Snippet
To_CSV	Writes DataFrame to a comma-separated values (CSV) file.	`df.to_csv('file.csv')`
To_Excel	Writes DataFrame to an Excel file.	`df.to_excel('file.xlsx')`
To_SQL	Writes DataFrame to a SQL database.	`df.to_sql('table_name', conn)`
To_JSON	Writes DataFrame to a JSON formatted string.	`df.to_json('file.json')`
To_HTML	Writes DataFrame to HTML tables.	`df.to_html('file.html')`
To_Clipboard	Writes DataFrame to the clipboard.	`df.to_clipboard()`

Create Test Objects

Actions	Description	Example Snippet
Dataframe	Constructs a DataFrame object.	`df = pd.DataFrame(data)`
Series	Constructs a Series object.	`s = pd.Series(data)`
Index	Constructs an Index object.	`index = pd.Index(data)`

DataFrame Basics

Actions	Description	Example Snippet
Return Dimensions of a DataFrame	Gets shape of DataFrame.	`df.shape`
Read CSV file into a DataFrame	Reads CSV and returns DataFrame object.	`df = pd.read_csv('file.csv')`
Return the data type of each column	Returns data types of columns in DataFrame.	`df.dtypes`

Selecting DataFrame Values

Actions	Description	Example Snippet
Select the rank column from f500	Selects a specific column from DataFrame.	`f500['rank']`
Select the first 3 rows from f500	Slices the DataFrame.	`f500.head(3)`

ILOC / LOC

Actions	Description	Example Snippet
LOC	Access a group of rows and columns by labels.	`df.loc[row_index, 'column_name']`
ILOC	Access a group of rows and columns by integer index.	`df.iloc[row_index, col_index]`

Referência rápida de funções do Pandas para manipulação de dados

Lembre-se

A limpeza de dados textuais é uma habilidade de **detetive**. A atenção aos detalhes e a verificação constante são essenciais para evitar erros graves na análise. Sempre desconfie dos dados textuais e use o `df.value_counts()` como seu principal aliado.

Encerramento e Desafio Final

O que Aprendemos Hoje

- Por que dados textuais são desafiadores e a importância da **limpeza de strings** para análises precisas
- Como usar o **acessor .str** do Pandas para manipular colunas de texto de forma eficiente
- Técnicas para **detectar e corrigir** inconsistências como variações de caixa, espaços invisíveis e erros de categoria
- Métodos para **filtrar, extrair e transformar** informações textuais em formatos padronizados
- A importância da **documentação** e organização do processo de limpeza de dados

> *"A limpeza de dados textuais é uma habilidade de detetive. A atenção aos detalhes evita erros graves na análise e pode ser a diferença entre uma conclusão correta e uma completamente equivocada."*

Na próxima aula, exploraremos técnicas avançadas de visualização de dados e como apresentar insights de forma eficaz.

Desafio Final

Encontre uma reportagem ou um gráfico em que o erro de formatação de um texto (uma cidade, um nome) poderia ter causado um problema na análise.

Instruções:

Pesquise em jornais, blogs de dados ou relatórios públicos
Identifique um caso onde a inconsistência textual gerou ou poderia gerar problemas
Explique qual seria a abordagem correta usando as técnicas aprendidas hoje
Compartilhe seu exemplo na próxima aula

Dica: Procure por gráficos com categorias duplicadas, relatórios com inconsistências em nomes de cidades/estados, ou análises onde a mesma entidade aparece com grafias diferentes.

Lembre-se Sempre

Sempre desconfie dos dados textuais e use o **df.value_counts()** como seu principal aliado na detecção de inconsistências. A qualidade da sua análise depende diretamente da qualidade da limpeza dos seus dados.