

## PPT 11 – Fundamentos de Python (UFCD 10793)

Sandra Liliana Meira de Oliveira



A análise de dados:

- Ajuda-nos a descobrir informação útil;
- Responder a questões;
- Prever o futuro

O pandas é uma biblioteca Python para análise e manipulação de dados, desenvolvida para simplificar tarefas que seriam complexas ou demoradas com ferramentas como listas ou dicionários em Python puro.

Baseia-se no NumPy, uma biblioteca de arrays numéricos de alta performance, e foi projetada para trabalhar com grandes volumes de dados, mantendo a flexibilidade e simplicidade.

## Porquê usar pandas?

- **Estruturas de Dados Ricas:** O pandas oferece **DataFrame** e **Series**, estruturas tabulares que são intuitivas para manipular dados etiquetados e heterogêneos.
- **Funcionalidades de Alto Nível:** Permite carregar, transformar e analisar dados rapidamente.
- **Integração:** Funciona bem com outras bibliotecas de Data Science (e.g., NumPy, Matplotlib, scikit-learn).
- **Eficiência:** É otimizado para manipulação de grandes volumes de dados em memória.

## Cenários Reais

- **Limpeza de Dados:** Identificar e corrigir valores ausentes ou inconsistentes.
- **Exploração de Dados:** Extrair estatísticas resumidas e identificar padrões.
- **Transformação:** Combinar e reorganizar dados de diferentes fontes.
- **Automação:** Automatizar relatórios e análises repetitivas.

## **Como instalar a biblioteca**

No terminal executa o comando

```
pip install pandas
```

# Estruturas de Dados Centrais: Series e DataFrame

## Series

- Uma **Series** é uma estrutura de dados unidimensional no pandas, similar a um array do NumPy ou uma lista em Python, mas com a vantagem de incluir um **índice** para cada elemento.
- Cada valor na Series é associado a uma **etiqueta única** (índice), que pode ser numérica ou textual. Isto facilita o acesso direto aos elementos através destes rótulos.

## Principais Características:

- **Homogeneidade:** Todos os elementos de uma Series são do mesmo tipo de dados.
- **Indexação:** Cada elemento tem um rótulo que permite acessá-lo de forma direta.
- **Versatilidade:** Pode ser criada a partir de listas, dicionários, arrays NumPy, ou mesmo valores escalares.

**Series**

	nome		idade	
Índice	0	Maria	0	25
	1	José	1	56
	2	Ana	2	31
	3	Paulo	3	43
				Linhas

# Estruturas de Dados Centrais: Series e DataFrame

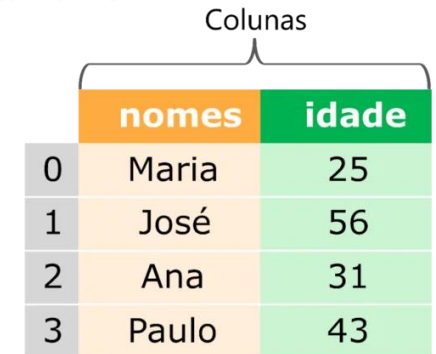
## DataFrame

- O **DataFrame** é uma estrutura bidimensional (tabela) que organiza os dados em linhas e colunas.
- Cada **coluna** de um DataFrame é uma **Series**, e todas as colunas partilham o mesmo índice (linhas).
- Permite trabalhar com dados heterogêneos (e.g., texto, números, datas).

## Principais Características:

- **Heterogeneidade:** Cada coluna pode ter um tipo de dado diferente.
- **Indexação dupla:** Linhas e colunas são indexadas, permitindo acesso fácil aos dados.
- **Flexibilidade:** Pode ser criado a partir de listas, dicionários, arrays NumPy, ou ficheiros como CSV e Excel e ainda através de consultas SQL.

**DataFrame**




	Colunas	
	nomes	idade
0	Maria	25
1	José	56
2	Ana	31
3	Paulo	43



# Exemplos – Series e DataFrame

## 1. Criar uma Series a partir de uma lista

python


 Copy code

```
import pandas as pd

# Criar uma Series simples
numeros = [10, 20, 30, 40]
serie_numeros = pd.Series(numeros)
print(serie_numeros)
```

Saída:

go

 Copy code

```
0    10
1    20
2    30
3    40
dtype: int64
```

## Explicação:

A Series atribui automaticamente índices (0, 1, 2, ...) a cada elemento.

# Exemplos – Series e DataFrame

## 2. Criar uma Series com índices personalizados

python

```
alunos = ['João', 'Maria', 'Pedro', 'Ana']
notas = [15, 18, 14, 19]

serie_notas = pd.Series(notas, index=alunos)
print(serie_notas)
```

Saída:

go

```
João    15
Maria   18
Pedro   14
Ana     19
dtype: int64
```

## Explicação:

- Os índices personalizados permitem aceder aos valores diretamente

```
print(serie_notas['Maria']) # Resultado: 18
```

# Exemplos – Series e DataFrame

## 3. Criar uma Series a partir de um dicionário

python

```
dados = {'Braga': 120, 'Porto': 250, 'Lisboa': 340}
serie_cidades = pd.Series(dados)
print(serie_cidades)
```

Saída:

go

```
Braga      120
Porto      250
Lisboa     340
dtype: int64
```

## Explicação:

- Os índices são as chaves

# Exemplos – Series e DataFrame

## Exemplos Práticos com DataFrame

### 1. Criar um DataFrame a partir de um dicionário

```
python

dados = {
    'Nome': ['João', 'Maria', 'Pedro', 'Ana'],
    'Idade': [25, 30, 22, 28],
    'Cidade': ['Braga', 'Porto', 'Lisboa', 'Coimbra']
}

df = pd.DataFrame(dados)
print(df)
```

### Saída:

```
markdown
```

	Nome	Idade	Cidade
0	João	25	Braga
1	Maria	30	Porto
2	Pedro	22	Lisboa
3	Ana	28	Coimbra

### Explicação:

- Cada coluna do DataFrame é uma Series.

```
python

print(df['Nome']) # Retorna a Series da coluna Nome
```

# Exemplos – Series e DataFrame

## 2. Criar um DataFrame a partir de uma lista de listas

python

```
dados = [  
    ['João', 25, 'Braga'],  
    ['Maria', 30, 'Porto'],  
    ['Pedro', 22, 'Lisboa']  
]  
  
df = pd.DataFrame(dados, columns=['Nome', 'Idade', 'Cidade'])  
print(df)
```

Saída:

markdown

	Nome	Idade	Cidade
0	João	25	Braga
1	Maria	30	Porto
2	Pedro	22	Lisboa

# Exemplos – Series e DataFrame

## 3. Carregar dados a partir de um ficheiro CSV

python

```
# Supondo um ficheiro chamado "dados.csv" com as colunas Nome, Idade, Cidade  
df = pd.read_csv('dados.csv')  
print(df.head()) # Visualizar as primeiras 5 linhas
```

### Explicação:

O pandas suporta a leitura de ficheiros CSV, Excel, SQL, entre outros.

Pode-se inspecionar rapidamente os dados carregados com funções como **.head()** e **.info()**.

# Operações Fundamentais com DataFrames

## 1. Selecionar uma coluna

```
python  
  
print(df['Nome'])
```

### Saída:

```
vbnet  
  
0      João  
1      Maria  
2      Pedro  
Name: Nome, dtype: object
```

## 2. Selecionar várias colunas

```
python  
  
print(df[['Nome', 'Idade']])
```

### Saída:

```
markdown  
  
      Nome  Idade  
0  João    25  
1  Maria    30  
2  Pedro    22
```

# Operações Fundamentais com DataFrames

## 3. Selecionar linhas por índice

python

*# Selecionar a primeira linha*

```
print(df.iloc[0])
```

*# Selecionar a linha onde Nome é "Maria"*

```
print(df[df['Nome'] == 'Maria'])
```



# Operações Fundamentais com DataFrames

## 4. Adicionar uma nova coluna

python

```
df['Profissão'] = ['Engenheiro', 'Médico', 'Estudante']  
print(df)
```

Saída:

markdown

	Nome	Idade	Cidade	Profissão
0	João	25	Braga	Engenheiro
1	Maria	30	Porto	Médico
2	Pedro	22	Lisboa	Estudante

# Operações Fundamentais com DataFrames

## 5. Filtrar dados com condições

python

```
# Filtrar linhas onde Idade é maior que 25  
print(df[df['Idade'] > 25])
```

### Saída:

markdown

	Nome	Idade	Cidade	Profissão
1	Maria	30	Porto	Médico

# Operações Fundamentais com DataFrames

Renomear índices das linhas de 0,1,2,... para um nome mais significativo – atributo `index`

```
>>> df.index = ['one', 'two', 'three']
>>> df
```

	date	name	price	shares	owner
one	2001-12-01	AA	12.3	100	Unknown
two	2012-02-10	IBM	10.3	30	Unknown
three	2010-04-09	GOOG	32.2	90	Unknown

Qualquer coluna do DataFrame pode ser definida como índice utilizando o atributo `set_index()`

```
>>> df = df.set_index(['name'])
>>> df
```

	date	price	shares	owner
name				
AA	2001-12-01	12.3	100	Unknown
IBM	2012-02-10	10.3	30	Unknown
GOOG	2010-04-09	32.2	90	Unknown

O índice pode ser reiniciado usando o comando `'reset_index'`.

# Operações Fundamentais com DataFrames

Qualquer coluna pode ser apagada usando os comandos `del` ou `drop`

```
>>> del df['owner']  
>>> df
```

	date	price	shares
name			
AA	2001-12-01	12.3	100
IBM	2012-02-10	10.3	30
GOOG	2010-04-09	32.2	90

```
>>> df.drop('shares', axis = 1)
```

	date	price
name		
AA	2001-12-01	12.3
IBM	2012-02-10	10.3
GOOG	2010-04-09	32.2

# Conclusão sobre Series e DataFrame

## 1. Series:

- Ideal para representar colunas individuais ou conjuntos de dados simples com rótulos.
- É frequentemente usada para manipular uma única dimensão de um dataset.

## 2. DataFrame:

- Estrutura mais rica e flexível, adequada para análises tabulares completas.
- Facilita operações como seleção, filtro e manipulação de múltiplas dimensões dos dados.
- Os dados podem ser acedidos de duas formas, ou seja, utilizando o índice de linha e coluns.

# Inspecção e Limpeza de Dados

A **qualidade dos dados** é fundamental para análises confiáveis. Antes de qualquer transformação, deve-se avaliar a integridade, completude e consistência do dataset.

## Processo de Inspecção

### 1. Estrutura e Dimensão:

- Quantas linhas e colunas o dataset tem? (**df.shape**)
- Quais são os tipos de dados em cada coluna? (**df.info()**)

### 2. Resumo Estatístico:

- Qual é a distribuição dos valores? (**df.describe()**)
- Existem outliers ou anomalias visíveis?

### 3. Valores Ausentes:

- Linhas ou colunas incompletas (**df.isnull().sum()**).

### 4. Coerência de Dados:

- Dados duplicados ou inconsistentes.
- Formato correto em colunas (e.g., datas, números).

# Inspeção e Limpeza de Dados

## Tratamento de Dados Ausentes

- **Remoção:** Eliminar linhas ou colunas onde faltam valores (**.dropna()**).
- **Substituição:** Preencher valores ausentes com médias, medianas, ou valores específicos (**.fillna()**).
- **Imputação Avançada:** Utilizar algoritmos para prever os valores ausentes.

## Exemplo Real:

Um dataset de vendas tem valores ausentes na coluna "Preço". Pode-se:

1. Substituir por 0 se o produto for gratuito.
2. Substituir pela média de preços.

# Manipulação e Transformação de Dados

A manipulação de dados envolve preparar datasets para análise, reorganizando-os, combinando informações de diferentes fontes, ou resumindo-os em métricas significativas.

## Ordenação

- Ordenar por uma ou mais colunas facilita a visualização de tendências ou identificação de extremos (e.g., cliente com maior volume de compras).

## Combinação de Dados

1. **Concatenação:** Combina datasets empilhando linhas ou colunas (`pd.concat()`).
2. **Junção:** Une datasets com base numa chave comum, semelhante a SQL (`pd.merge()`).



# Manipulação e Transformação de Dados

## Agrupamento e Agregação

- O método `.groupby()` permite dividir os dados em grupos com base em valores de uma coluna, aplicar uma função de agregação (e.g., `sum`, `mean`) e obter resultados agrupados.

### Exemplo Prático:

Calcular o total de vendas por região: **`df.groupby('Região')['Vendas'].sum()`**

## Transformações Avançadas

- Aplicar funções personalizadas a colunas com **`.apply()`**.
- Criar novas colunas derivadas de outras.

# Datas, Strings e Funcionalidades Mais Avançadas

**Manipulação de Datas:** Datas são frequentemente usadas em análises temporais (e.g., vendas mensais). No pandas, as colunas de texto podem ser convertidas em objetos datetime para extração de componentes como ano, mês, dia, ou cálculos de intervalos.

## Exemplo:

- Coluna de texto "2023-12-20" → Objeto datetime.
- Extração: Ano = 2023, Mês = 12.

**Manipulação de Strings:** Colunas com texto requerem frequentemente limpeza (e.g., remover espaços, substituir caracteres). O pandas oferece métodos para manipular strings diretamente.

## Exemplo:

- Substituir "n/d" por "Não disponível".
- Converter para minúsculas: `df['Coluna'].str.lower()`.

# Análise Exploratória e Visualização com pandas

A análise exploratória de dados (EDA) ajuda a:

1. **Entender a Distribuição:** Quais são os valores mais comuns?  
Existem valores atípicos?
2. **Identificar Relações:** Como variáveis se relacionam (e.g., correlação entre preço e vendas).

## Visualização com pandas

- Embora o pandas não seja especializado em gráficos, fornece métodos simples para criar histogramas, gráficos de barras e boxplots diretamente do DataFrame.

### Exemplo Prático:

Visualizar a distribuição de idades:

```
python  
  
df['Idade'].plot(kind='hist')
```

Limpeza, Tratamento, Manipulação e Transformação de  
Dados com Pandas – análise de funcionalidades específicas

# Importar e Exportar Dados

**Importar Dados**

Formatos csv, json, excel, sql

**Exportar Dados**

# Formato csv

- O que é o formato csv?
- Que outros formatos se podem usar para guardar dados??

```
3,?,alfa-romero,gas,std,two,convertible,rwd,front,88.60,168.80,64.10,48.80,2548,dohc,four,130,mpfi,3.47,2.68,9.00,111,5000,21,27,13495
3,?,alfa-romero,gas,std,two,convertible,rwd,front,88.60,168.80,64.10,48.80,2548,dohc,four,130,mpfi,3.47,2.68,9.00,111,5000,21,27,16500
1,?,alfa-romero,gas,std,two,hatchback,rwd,front,94.50,171.20,65.50,52.40,2823,ohcv,six,152,mpfi,2.68,3.47,9.00,154,5000,19,26,16500
2,164,audi,gas,std,four,sedan,fwd,front,99.80,176.60,66.20,54.30,2337,ohc,four,109,mpfi,3.19,3.40,10.00,102,5500,24,30,13950
2,164,audi,gas,std,four,sedan,4wd,front,99.40,176.60,66.40,54.30,2824,ohc,five,136,mpfi,3.19,3.40,8.00,115,5500,18,22,17450
2,?,audi,gas,std,two,sedan,fwd,front,99.80,177.30,66.30,53.10,2507,ohc,five,136,mpfi,3.19,3.40,8.50,110,5500,19,25,15250
1,158,audi,gas,std,four,sedan,fwd,front,105.80,192.70,71.40,55.70,2844,ohc,five,136,mpfi,3.19,3.40,8.50,110,5500,19,25,17710
1,?,audi,gas,std,four,wagon,fwd,front,105.80,192.70,71.40,55.70,2954,ohc,five,136,mpfi,3.19,3.40,8.50,110,5500,19,25,18920
1,158,audi,gas,turbo,four,sedan,fwd,front,105.80,192.70,71.40,55.90,3086,ohc,five,131,mpfi,3.13,3.40,8.30,140,5500,17,20,23875
0,?,audi,gas,turbo,two,hatchback,4wd,front,99.50,178.20,67.90,52.00,3053,ohc,five,131,mpfi,3.13,3.40,7.00,160,5500,16,22,?
2,192,bmw,gas,std,two,sedan,rwd,front,101.20,176.80,64.80,54.30,2395,ohc,four,108,mpfi,3.50,2.80,8.80,101,5800,23,29,16430
0,192,bmw,gas,std,four,sedan,rwd,front,101.20,176.80,64.80,54.30,2395,ohc,four,108,mpfi,3.50,2.80,8.80,101,5800,23,29,16925
0,188,bmw,gas,std,two,sedan,rwd,front,101.20,176.80,64.80,54.30,2710,ohc,six,164,mpfi,3.31,3.19,9.00,121,4250,21,28,20970
0,188,bmw,gas,std,four,sedan,rwd,front,101.20,176.80,64.80,54.30,2765,ohc,six,164,mpfi,3.31,3.19,9.00,121,4250,21,28,21105
1,?,bmw,gas,std,four,sedan,rwd,front,103.50,189.00,66.90,55.70,3055,ohc,six,164,mpfi,3.31,3.19,9.00,121,4250,20,25,24565
0,?,bmw,gas,std,four,sedan,rwd,front,103.50,189.00,66.90,55.70,3230,ohc,six,209,mpfi,3.62,3.39,8.00,182,5400,16,22,30760
0,?,bmw,gas,std,two,sedan,rwd,front,103.50,193.80,67.90,53.70,3380,ohc,six,209,mpfi,3.62,3.39,8.00,182,5400,16,22,41315
0,?,bmw,gas,std,four,sedan,rwd,front,110.00,197.00,70.90,56.30,3505,ohc,six,209,mpfi,3.62,3.39,8.00,182,5400,15,20,36880
2,121,chevrolet,gas,std,two,hatchback,fwd,front,88.40,141.10,60.30,53.20,1488,1,three,61,2bbl,2.91,3.03,9.50,48,5100,47,53,5151
```

Output:

Original file:

1	A	CSE
0	2	B Commerce
1	3	C CSE
2	4	D Humanities
3	5	E Commerce
4	6	F CSE

Modified file:

	id	name	profession
0	2	B	Commerce
1	3	C	CSE
2	4	D	Humanities
3	5	E	Commerce
4	6	F	CSE

??headers????

??Atributos (nome, intervalo)????

# Leitura de Ficheiros

- `read_csv`: ler os dados a partir do ficheiro CSV.
- `index_col = None`: não há índice, ou seja, a primeira coluna são dados (não tem cabeçalhos)
- `head()`: mostrar apenas os primeiros cinco elementos do DataFrame
- `tail()`: mostrar apenas os últimos cinco elementos do DataFrame

Se houver algum erro ao ler o ficheiro devido à codificação, devemos tente seguir a opção:

```
titles = pd.read_csv('titles.csv', index_col=None, encoding='utf-8')
```

```
>>> import pandas as pd

>>> casts = pd.read_csv('cast.csv', index_col=None)
>>> casts.head()
   title  year  name  type  character  n
0  Closet Monster  2015  Buffy #1  actor  Buffy 4  31.0
1  Suuri illusioni  1985  Homo $  actor  Guests  22.0
2  Battle of the Sexes  2017  $hutter  actor  Bobby Riggs Fan  10.0
3  Secret in Their Eyes  2015  $hutter  actor  2002 Dodger Fan  NaN
4  Steve Jobs  2015  $hutter  actor  1988 Opera House Patron  NaN

>>> titles = pd.read_csv('titles.csv', index_col=None)
>>> titles.tail()
   title  year
49995  Rebel  1970
49996  Suzanne  1996
49997  Bomba  2013
49998  Aao Jao Ghar Tumhara  1984
49999  Mrs. Munck  1995
```

# Análise dos Dados no Dataframe

Se simplesmente escrevermos o nome do serão mostrados as primeiras trinta e as últimas vinte linhas do ficheiro juntamente com a lista completa de colunas.

```
>>> pd.set_option('max_rows', 10, 'max_columns', 10)
```

```
>>> titles
```

	title	year
0	The Rising Son	1990
1	The Thousand Plane Raid	1969
2	Crucea de piatra	1993
3	Country	2000
4	Gaiking II	2011
...	...	...
49995	Rebel	1970
49996	Suzanne	1996
49997	Bomba	2013
49998	Aao Jao Ghar Tumhara	1984
49999	Mrs. Munck	1995

```
[50000 rows x 2 columns]
```

**len** : O commando 'len' pode ser usada para ver o número total de linhas no ficheiro.

```
>>> len(titles)
50000
```

Os comandos **head()** e **tail ()** podem ser usados para nos lembrarmos do cabeçalho e do conteúdo do ficheiro. Estes dois comandos mostrarão as primeiras e últimas 5 linhas, respetivamente, do ficheiro. Além disso, podemos alterar o total número de linhas a exibir

```
>>> titles.head(3)
```

	title	year
0	The Rising Son	1990
1	The Thousand Plane Raid	1969
2	Crucea de piatra	1993



# Seleção de Linhas e Colunas num Dataframe

Qualquer linha ou coluna do DataFrame pode ser selecionada passando o nome da coluna ou linhas.

Depois de selecionar linha ou coluna a partir do DataFrame, torna-se unidimensional, portanto é considerado como Série.

**loc**: usar o comando 'loc' ou 'iloc' para selecionar uma linha a partir do DataFrame

```
>>> t = titles['title']
```

```
>>> type(t)
<class 'pandas.core.series.Series'>
```

```
>>> t.head()
0          The Rising Son
1  The Thousand Plane Raid
2      Crucea de piatra
```

# Filtrar Dados

Os dados podem ser filtrados fornecendo alguma expressão booleana no DataFrame.

Por exemplo, no código ao lado, os filmes que foram lançados após 1985 são filtrados a partir do nome das colunas do DataFrame e armazenados num novo DataFrame (after85)

```
>>> # display movie in years 1990 - 1999
>>> t = titles
>>> movies90 = t[ (t['year']>=1990) & (t['year']<2000) ]
>>> movies90.head()
```

	title	year
0	The Rising Son	1990
2	Crucea de piatra	1993
12	Poka Makorer Ghar Bosoti	1996
19	Maa Durga Shakti	1999
24	Conflict of Interest	1993

```
>>>
```

```
>>> # movies after 1985
>>> after85 = titles[titles['year'] > 1985]
>>> after85.head()
```

	title	year
0	The Rising Son	1990
2	Crucea de piatra	1993
3	Country	2000
4	Gaiking II	2011
5	Medusa (IV)	2015

```
>>>
```

# Ordenar Dados

Para ordenarmos os dados usamos `sort_index` ou `sort_values`

```
>>> # find all movies named as 'Macbeth'
>>> t = titles
>>> macbeth = t[ t['title'] == 'Macbeth']
>>> macbeth.head()
      title  year
```

```
>>> # by default, sort by index i.e. row header
>>> macbeth = t[ t['title'] == 'Macbeth'].sort_index()
>>> macbeth.head()
```

```
>>> # sort by year
>>> macbeth = t[ t['title'] == 'Macbeth'].sort_values('year')
>>> macbeth.head()
```

4226	Macbeth	1913
9322	Macbeth	2006
11722	Macbeth	2013
17166	Macbeth	1997
25847	Macbeth	1998

	title	year
4226	Macbeth	1913
9322	Macbeth	2006
11722	Macbeth	2013
17166	Macbeth	1997
25847	Macbeth	1998

```
>>>
```

	title	year
4226	Macbeth	1913
17166	Macbeth	1997
25847	Macbeth	1998
9322	Macbeth	2006
11722	Macbeth	2013

```
...
```

# Valores Nulos

várias colunas podem não conter valores, que são geralmente preenchidos como NaN.

	title	year	name	type	character	n
3	Secret in Their Eyes	2015	\$hutter	actor	2002 Dodger Fan	NaN
4	Steve Jobs	2015	\$hutter	actor	1988 Opera House Patron	NaN

---

Estes valores nulos podem ser facilmente selecionados, não selecionados ou o seu conteúdo pode ser substituído por quaisquer outros valores, por exemplo, strings vazias ou 0

O comando `'isnull'` devolve verdadeiro se a linha tem algum valor nulo

```
>>> c = casts
>>> c['n'].isnull().head()
0    False
1    False
2    False
3     True
4     True
Name: n, dtype: bool
```

O comando `'notnull'` é o oposto do anterior

```
>>> c['n'].notnull().head()
0     True
1     True
2     True
3    False
4    False
Name: n, dtype: bool
```

# Valores Nulos

Para exibir as linhas com valores nulos, a condição deve ser passada no DataFrame,

```
>>> c[c['n'].isnull()].head(3)
```

	title	year	name	type	character	n
3	Secret in Their Eyes	2015	\$hutter	actor	2002 Dodger Fan	NaN
4	Steve Jobs	2015	\$hutter	actor	1988 Opera House Patron	NaN
5	Straight Outta Compton	2015	\$hutter	actor	Club Patron	NaN

```
>>>
```

Os valores NaN podem ser preenchidos utilizando **fillna**, **ffill** (preenchimento dianteiro) e **bfill** (preenchimento para trás) etc.

Com **fillna** os valores 'NaN' são substituídos por **NA**

```
>>> c_fill = c[c['n'].isnull()].fillna('NA')
```

```
>>> c_fill.head(2)
```

	title	year	name	type	character	n
3	Secret in Their Eyes	2015	\$hutter	actor	2002 Dodger Fan	NA
4	Steve Jobs	2015	\$hutter	actor	1988 Opera House Patron	NA

# Operações com Strings

Podem ser realizadas várias operações sobre as strings conteúdo do Datafra, utilizando a opção '.str'.

Vamos, por exemplo, procurar o filme "Maa"

```
>>> t = titles
>>> t[t['title'] == 'Maa']
      title  year
38880   Maa  1968
>>>
```

Agora, queremos pesquisar todos os filmes que começam com "Maa".

```
>>> t[t['title'].str.startswith("Maa ")] .head(3)
      title  year
19   Maa Durga Shakti  1999
3046   Maa Aur Mamta  1970
7470  Maa Vaibhav Laxmi  1989
>>>
```

# Contar Valores

O número total de ocorrências pode ser contado com a opção "value\_counts()". No seguinte código, o número total de filmes é determinado com base na coluna 'year'

```
>>> t['year'].value_counts().head()
2016      2363
2017      2138
2015      1849
2014      1701
2013      1609
Name: year, dtype: int64
```

# Group By – Nome das Colunas

Os dados podem ser agrupados por cabeçalhos de colunas.

O comando 'groupby' devolve um objeto, e normalmente precisamos de uma funcionalidade extra para obter alguns resultados.

Por exemplo, no código abaixo, os dados são agrupados pela coluna 'year' e depois usa-se o comando `size`.

A opção de `size()` conta o número total de linhas para cada ano; portanto, o resultado do código seguinte é o mesmo que o comando '`count_values()`'.

```
>>> cg = c.groupby(['year']).size()
>>> cg.plot()
<matplotlib.axes._subplots.AxesSubplot object at 0xa9f14b4c>
>>> plt.show()
>>>
```



# Goup By – Nome das Colunas

A opção groupby pode tomar vários parâmetros.

Por exemplo, queremos agrupar o filmes do ator 'Aaron Abrams' baseados no ano:

```
>>> c = casts
>>> cf = c[c['name'] == 'Aaron Abrams']
>>> cf.groupby(['year']).size().head()
```

```
year
2003    2
2004    2
2005    2
2006    1
2007    2
dtype: int64
>>>
```

Queremos também ver a lista de filmes, então podemos passar dois parâmetros:

```
>>> cf.groupby(['year', 'title']).size().head()
year  title
2003  The In-Laws      1
      The Visual Bible: The Gospel of John  1
2004  Resident Evil: Apocalypse      1
      Siblings      1
2005  Cinderella Man      1
dtype: int64
>>>
```

# Goup By – Nome das Colunas

No código anterior, a operação groupby é realizada em primeiro no 'year' e depois no 'title'. Ou seja, primeiro todos os filmes são agrupados por ano. Depois disso, o resultado deste grupo é novamente agrupado com base em títulos.

Nota que, o primeiro comando do grupo organizou o ano por ordem ou seja, 2003, 2004 e 2005, etc.; em seguida, o comando seguinte organizou o título por ordem alfabética.

Em seguida, queremos agrupar com base nas classificações máximas num ano; ou seja, queremos agrupar os itens por ano e ver a classificação máxima nesses anos

```
>>> c.groupby(['year']).n.max().head()
year
1912      6.0
1913     14.0
1914     39.0
1915     14.0
1916     35.0
Name: n, dtype: float64
```

Da mesma forma, podemos verificar a classificação mínima,

```
>>> c.groupby(['year']).n.min().head()
year
1912      6.0
1913      1.0
1914      1.0
1915      1.0
1916      1.0
Name: n, dtype: float64
```

Por último, queremos verificar a classificação média de cada ano

```
>>> c.groupby(['year']).n.mean().head()
year
1912      6.000000
1913      4.142857
1914      7.085106
1915      4.236111
1916      5.037736
Name: n, dtype: float64
```

# Função Plot do Pandas

Embora o pandas não seja especializado em gráficos, fornece métodos simples para criar histogramas, gráficos de barras e boxplots diretamente do DataFrame.

O parâmetro kind no método plot() do pandas define o tipo de gráfico que será gerado.

```
df['coluna'].plot(kind='line')
```

Tipo (kind)	Gráfico	Usado Para
line	Gráfico de linha	Dados contínuos ou séries temporais
bar	Gráfico de barras verticais	Comparar categorias
barh	Gráfico de barras horizontais	Comparar categorias (horizontal)
hist	Histograma	Distribuição de variáveis numéricas
box	Boxplot	Resumo estatístico e outliers
kde / density	Densidade	Densidade de probabilidade
area	Gráfico de área	Dados contínuos (áreas preenchidas)
pie	Gráfico de pizza	Proporções ou percentagens
scatter	Gráfico de dispersão	Relação entre duas variáveis
hexbin	Gráfico hexagonal	Densidade bidimensional (grandes datasets)

Considerações Importantes acerca de ...  
Limpeza de Dados e AED

# Limpeza de Dados e AED

Antes de se efetuar a aplicação de técnicas estatísticas aos nossos dados é necessário:

- Limpar os dados;
- Efetuar uma análise exploratória de dados.

A **Análise Exploratória de Dados** começa **antes** da aplicação de **qualquer** técnica **estatística**. É necessário **conhecer** os dados que serão analisados, se eles estão no **formato** correto, se possuem algum **relacionamento**, o significado **exato** de cada **coluna** que será **utilizada** da tabela de dados que será analisada....

# Limpeza de Dados e AED

- Os cientistas de dados passam grande parte do seu tempo a limpar conjuntos de dados e a transformá-los em algo com valor.
- Na verdade, os primeiros passos de obtenção e limpeza de dados constituem 80% do trabalho.

# Limpeza de Dados e AED

A estatística **descritiva** é a etapa **inicial** da análise utilizada para **descrever** e **resumir** os dados. A disponibilidade de uma **grande quantidade de dados** e de métodos **computacionais** muito eficientes revigorou esta área da **estatística**.

A **Análise Exploratória de Dados** começa **antes** da aplicação de **qualquer** técnica **estatística**.

É necessário **conhecer** os dados que serão analisados, se eles estão no **formato** correto, se possuem algum **relacionamento**, o significado **exato** de cada **coluna** que será **utilizada** da tabela de dados que será analisada.

**Após** a recolha e a importação dos dados para uma base de dados, o próximo passo é a **análise descritiva**.

Esta etapa é fundamental, pois uma análise descritiva detalhada permite ao pesquisador familiarizar-se com os dados, organizá-los e sintetizá-los de forma a obter as informações necessárias do **conjunto** de dados para responder as questões que estão sendo estudadas.

A Análise Exploratória de Dados vai **além** do uso **descritivo** da estatística, pois procura olhar de forma mais **profunda os dados**, sem **resumir** muito a **quantidade de informações**, utilizando uma diversidade de ferramentas computacionais para analisar os dados.

# Limpeza de Dados e AED

Devemos ser capazes de lidar com dados:

- não estruturados ...
- ... confusos, quer isso signifique valores em falta, formatação inconsistente, registos mal formados ou *outliers* sem sentido.



Deixar cair colunas desnecessárias

Alterar o índices

Utilizar métodos para limpar colunas

Utilizar a funções para limpar todo o conjunto de dados, em termos de elementos

Rebatizar colunas para um conjunto de rótulos mais reconhecível

Saltar/Não utilizar linhas desnecessárias

...



# Limpeza de Dados e AED

## AED:

- Mostrar dados (head() e tail()) ...
- Dimensão dos dados
- Tipos de Dados
- Trabalhar Valores NULOS
- Encontrar e Eliminar Duplicados
- Sumário estatístico (MAX, MIN, MEAN, ... describe())
- Agrupar dados
- Correlações entre variáveis
- Visualização de Dados - gráficos

# Limpeza de Dados e AED

Um estudo completo de limpeza de dados usando pandas inclui várias etapas:

1. **Exploração dos Dados**: A primeira etapa é explorar os dados para entender a sua estrutura, tipos de variáveis e a presença de valores ausentes ou duplicados. Isso pode ser feito usando funções do pandas como `head()`, `info()`, `describe()` e `value_counts()`.
2. **Tratamento de Valores Ausentes**: Após entender os dados, a próxima etapa é tratar os valores ausentes. Isso pode ser feito excluindo os valores ausentes, substituindo-os por um valor específico ou imputando os valores ausentes com um método estatístico como média, mediana ou moda. É importante considerar o efeito de excluir ou imputar valores ausentes na análise ou aplicação para a qual os dados serão utilizados.
3. **Tratamento de Valores Duplicados**: Após tratar os valores ausentes, a próxima etapa é tratar os valores duplicados. Isso pode ser feito excluindo os valores duplicados ou mantendo somente uma cópia de cada valor duplicado.
4. **Preparação das Variáveis**: Depois de tratar os valores ausentes e duplicados, a próxima etapa é preparar as variáveis para a análise ou aplicação. Isso pode incluir trabalhar com variáveis categóricas, normalizando variáveis numéricas ou criando novas variáveis a partir das existentes.
5. **Exportar os Dados Limpos**: Finalmente, é importante exportar os dados limpos para um formato apropriado para a análise ou aplicação, como CSV ou Excel.

# Limpeza de Dados e AED

Um procedimento para realizar uma limpeza de dados **usando pandas** inclui os seguintes passos:

1. **Importar os dados**: Use a função `pd.read_csv()` ou `pd.read_excel()` para importar os dados de um arquivo CSV ou Excel, respectivamente.
2. **Analisar as informações gerais dos dados**: Utilizar as funções `head()`, `info()`, `describe()` e `value_counts()` para obter uma visão geral dos dados e identificar problemas como valores ausentes, tipos de variáveis incorretos e outliers.
3. **Tratar valores ausentes**: Utilizar as funções `dropna()` ou `fillna()` para remover ou preencher valores ausentes, respectivamente.
4. **Tratar valores duplicados**: Utilizar a função `drop_duplicates()` para remover valores duplicados.
5. **Renomear colunas**: Utilizar a função `rename()` para renomear colunas para nomes mais claros e significativos.
6. **Converter tipos de variáveis**: Utilizar as funções `astype()` ou `to_datetime()` para converter tipos de variáveis como strings para inteiros, flutuantes ou datas.
7. **Filtrar linhas**: Selecionar apenas as linhas que atendem a certas condições.
8. **Agrupar e resumir dados**: Utilizar a função `groupby()` para agrupar dados por uma ou mais colunas e aplicar funções de agregação como `mean()`, `sum()` ou `count()`.
9. **Salvar dados limpos**: Utilize as funções `to_csv()` ou `to_excel()` para salvar os dados limpos

# Limpeza de Dados e AED

## Técnicas para tratar valores ausentes

Existem várias técnicas para tratar valores ausentes em dados, cada uma com suas próprias vantagens e desvantagens. Algumas técnicas comuns incluem:

1. Remover as linhas ou colunas que contêm valores ausentes: Essa técnica é útil se os valores ausentes representarem uma pequena percentagem dos dados e não fornecerem informações importantes.
2. Substituir os valores ausentes com a média, mediana ou moda: Essa técnica é útil quando os valores ausentes são distribuídos de forma aleatória e não têm uma distribuição assimétrica.
3. Utilizar algoritmos de machine learning para prever os valores ausentes: Essa técnica é útil quando existem relações complexas entre as variáveis e a quantidade de dados ausentes é grande.
4. Utilizar técnicas de imputação estatística como imputação múltipla ou imputação por regressão.

Em geral, a escolha da técnica de tratamento de valores ausentes dependerá do conjunto de dados e do objetivo do estudo. É importante realizar uma avaliação dos dados e do impacto dos valores ausentes no modelo e resultado final.