

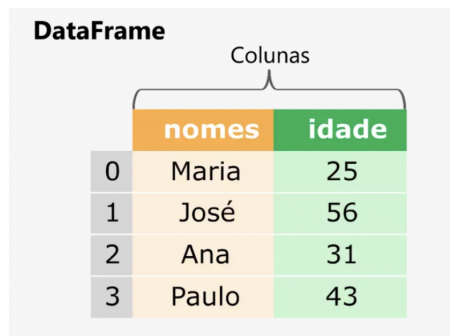
Tarefa 8

Professores:	Cristiana Neto, Pedro Oliveira, Vitor Alves
Disciplina:	Linguagens para Computação Numérica
Tema:	Pandas
Data:	Abril de 2022

1 Introdução

Na aula anterior abordamos a manipulação de dados a partir de ficheiros. Adicionalmente, existem várias bibliotecas em Python que fornecem ferramentas para análise e manipulação de dados. Nesta aula vamos abordar a biblioteca open-source e de uso gratuito Pandas para construir tabelas a partir de um conjunto de dados. Esta biblioteca é bastante popular pois consegue ler, manipular, agregar e exibir os dados. Tudo isto é possível devido à estrutura primária do Pandas: os **DataFrames**.

O **DataFrame** é uma estrutura de dados tabular, semelhante a um folha de dados do Excel, em que tanto as linhas quanto as colunas apresentam rótulos.



O diagrama ilustra a estrutura de um DataFrame. No topo, o título "DataFrame" está à esquerda. À direita, o rótulo "Colunas" aponta para duas colunas: "nomes" (destacada em laranja) e "idade" (destacada em verde). Abaixo, há uma tabela com quatro linhas. A primeira coluna contém índices numéricos (0, 1, 2, 3) e as seguintes colunas contêm os dados correspondentes às colunas "nomes" e "idade".

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

A partir dos objetos principais a biblioteca Pandas disponibiliza um conjunto de funcionalidades sofisticadas de indexação, que permite reformatar, manipular, agregar ou selecionar subconjuntos específicos dos dados que estamos a trabalhar.

Assim, as principais operações de manipulação de dados efetuadas a partir do Pandas são:

1. Importar e/ou aceder a dados.
2. Ordenar os registos da tabela.
3. Selecionar e fatiar nos índices/eixos.
4. Filtrar registos.
5. Renomear os índices/eixos.
6. Modificar a disposição do conteúdo.
7. Modificar/transformar o conteúdo.
8. Aplicar funções/calcular medidas resumo.

9. Agregar por categorias e aplicar.
10. Concatenar tabelas.
11. Juntar ou conciliar tabelas.

Nesta aula vamos introduzir algumas destas operações, sendo que podem consultar a documentação do Pandas. para uma explicação mais detalhada de todos os seus métodos e funções.

A utilização deste biblioteca, naturalmente, requer a sua importação no código Python:

```
import pandas as pd
```

Uma nova tabela pode ser criada com `pandas.DataFrame()`. Como exemplo, vamos ilustrar a criação de uma tabela com os dados da Taxa bruta de Natalidade e da Taxa Bruta de Mortalidade, dos anos mais recentes. Estas taxas dizem-nos quantos bebés nasceram ou quantos óbitos foram registados por 1000 habitantes.

As tabelas estão organizadas por linhas e colunas. Vamos organizar a informação em três colunas:

1. A primeira coluna ('Ano') refere-se ao ano
2. A segunda coluna tem a correspondente taxa bruta de natalidade ('Natalidade')
3. A terceira coluna tem a correspondente taxa bruta de mortalidade ('Mortalidade').

```
In [1]: import pandas

população = pandas.DataFrame({
    'Ano': [ 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018 ],
    'Natalidade': [ 9.2, 8.5, 7.9, 7.9, 8.3, 8.4, 8.4, 8.5 ],
    'Mortalidade': [ 9.7, 10.2, 10.2, 10.1, 10.5, 10.7, 10.7, 11.0 ]
})
```

Com a tabela criada, são várias as ações que podemos realizar.

1.1 Consultar Tabela

Basta escrever `população` (variável atribuída ao DataFrame) e é-nos apresentado o conteúdo da tabela. Neste exemplo, a visualização funciona bem, porque a tabela é muito pequena.

```
In [2]: população
```

```
Out[2]:
```

	Ano	Natalidade	Mortalidade
0	2011	9.2	9.7
1	2012	8.5	10.2
2	2013	7.9	10.2
3	2014	7.9	10.1
4	2015	8.3	10.5
5	2016	8.4	10.7
6	2017	8.4	10.7
7	2018	8.5	11.0

Geralmente as tabelas são grande e será útil visualizar ou processar apenas uma parte de uma tabela. Desta forma, existe várias possibilidades para extrair apenas uma parte da tabela.

1.1.1 Primeiras Linhas

O método `head()` retorna apenas as primeiras linhas da tabela. Ou seja, usando a tabela anterior, usariamos `população.head(3)` para ver as primeiras 3 linhas. Se não for indicado um valor, o método `head()` apresenta as primeiras 5 linhas.

```
In [3]: população.head()
```

```
Out[3]:
```

	Ano	Natalidade	Mortalidade
0	2011	9.2	9.7
1	2012	8.5	10.2
2	2013	7.9	10.2
3	2014	7.9	10.1
4	2015	8.3	10.5

1.1.2 Últimas Linhas

O método `tail()` funciona de modo semelhante ao anterior, mas apresenta as últimas linhas da tabela. Pode ter como argumento o número de linhas que se pretende visualizar. Ou seja, `população.tail(1)` mostra apenas a última linha da tabela que criámos.

1.1.3 Por Colunas

Continuando a usar a tabela anteriormente criada, podemos mostrar apenas uma ou mais colunas, com as seguintes formas:

- `população.Natalidade`, só a coluna Natalidade
- `população['Natalidade']`, igual ao anterior
- `população[['Natalidade', 'Ano']]`, para apresentar duas colunas específicas

1.1.4 Por Posições

Podemos consultar os valores em posições específicas da tabela com o método `iloc()`. Exemplos:

```
# Selecionar a quinta linha
população.iloc[4]
# Selecionar a quinta e sexta linha
população.iloc[4:6]
# Selecionar as 5 primeiras linhas (linhas 0 a 4)
população.iloc[0:5]
#ou
população.head()
# Primeira linha, segunda coluna
população.iloc[0,1]
# As primeiras 3 linhas (linhas 0, 1 e 2) e as primeiras duas colunas (colunas 0 e 1)
população.iloc[0:3, 0:2]
```

1.1.5 Metainformação

Os métodos anteriores permitem-nos consultar a informação que está contida na tabela. Ou seja, o seu conteúdo. A metainformação dá-nos as propriedades da tabela e não o conteúdo propriamente dito. Exemplos:

✓ [2] população.shape #retorna o nr de linhas e colunas da tabela

0 s

(8, 3)

✓ [3] len(população) #retorna o nr de linhas da tabela

0 s

8

✓ [4] população.columns #retorna informação sobre as colunas da tabela

0 s

Index(['Ano', 'Natalidade', 'Mortalidade'], dtype='object')

✓ [5] população.dtypes #retorna o tipo de dados de cada coluna da tabela

0 s

```
Ano                int64
Natalidade         float64
Mortalidade         float64
dtype: object
```

[7] população.describe() #retorna algumas estatísticas

	Ano	Natalidade	Mortalidade
count	8.00000	8.000000	8.000000
mean	2014.50000	8.387500	10.387500
std	2.44949	0.408613	0.415546
min	2011.00000	7.900000	9.700000
25%	2012.75000	8.200000	10.175000
50%	2014.50000	8.400000	10.350000
75%	2016.25000	8.500000	10.700000
max	2018.00000	9.200000	11.000000



1.1.6 Valores agregados

É possível obter alguns valores agregados da tabela. No próximo exemplo, é retornado o máximo de cada coluna da tabela.

```
população.max()
```

```
Ano                2018.0
Natalidade          9.2
Mortalidade        11.0
dtype: float64
```

É possível ainda extrair valores agregados de uma coluna. No exemplo seguinte são extraídas a soma, a média, o máximo e o mínimo da coluna Natalidade.

```
população.Natalidade.sum()
```

67.1

```
população.Natalidade.mean()
```

8.3875

```
população.Natalidade.max()
```

9.2

```
população.Natalidade.min()
```

7.9

1.2 Ordenar Tabela

O método `sort_values` consegue ordenar a tabela tendo uma coluna como referência. Exemplos:

```
[8] #ordenar a tabela por linhas, mas usando a coluna 'Natalidade',  
população.sort_values(by=['Natalidade'])
```

	Ano	Natalidade	Mortalidade
2	2013	7.9	10.2
3	2014	7.9	10.1
4	2015	8.3	10.5
5	2016	8.4	10.7
6	2017	8.4	10.7
1	2012	8.5	10.2
7	2018	8.5	11.0
0	2011	9.2	9.7

```
[9] #A ordenação por ordem decrescente faz-se adicionando o parâmetro ascending=False  
população.sort_values(by=['Natalidade'], ascending=False)
```

	Ano	Natalidade	Mortalidade
0	2011	9.2	9.7
1	2012	8.5	10.2
7	2018	8.5	11.0
5	2016	8.4	10.7
6	2017	8.4	10.7
4	2015	8.3	10.5
2	2013	7.9	10.2
3	2014	7.9	10.1

Por sua vez, o método `sort_index()` permite ordenar pelo índice das linhas (`axis=0`) ou pelo índice das colunas (`axis=1`)

```
[12] #apresentar a coluna 'Natalidade' em primeiro lugar
população.sort_index(axis=1, ascending=False)
```

	Natalidade	Mortalidade	Ano
0	9.2	9.7	2011
1	8.5	10.2	2012
2	7.9	10.2	2013
3	7.9	10.1	2014
4	8.3	10.5	2015
5	8.4	10.7	2016
6	8.4	10.7	2017
7	8.5	11.0	2018



1.3 Filtrar Tabela

Para além das variadas consultas já apresentadas, é muito prático filtrar o conteúdo da tabela com base em expressões sobre os valores. Exemplos:

```
#apresentar as linhas com ano igual ou maior a 2015
população[ população.Ano >= 2015 ]

#apresentar as linhas com ano igual a 2012 ou igual a 2016
população[ população.Ano.isin( [ 2012, 2016 ] ) ]

#apresentar as linhas onde a natalidade é maior ou igual a 8 e a mortalidade menor ou igual 10
população[ (população.Natalidade >= 8.0) & (população.Mortalidade <= 10.0) ]
```

1.4 Modificar Tabela

Existem várias modificações que podemos aplicar à nossa tabela.

1.4.1 Renomeação

Podemos alterar os nomes das nossas colunas bem como dos indexes das nossas linhas. Exemplo:

```
[11] # Renomeia índices de coluna (variáveis)
população = população.rename(columns = {"Ano": "year", "Natalidade": "birth", "Mortalidade": "mortality"})
população
```

	year	birth	mortality
0	2011	9.2	9.7
1	2012	8.5	10.2
2	2013	7.9	10.2
3	2014	7.9	10.1
4	2015	8.3	10.5
5	2016	8.4	10.7
6	2017	8.4	10.7
7	2018	8.5	11.0

```
[9] # Renomeia índices de linhas (index)
população = população.rename(index = {0: "11", 1: "12", 2: "13", 3: "14", 4: "15", 5: "16", 6: "17", 7: "18"})
população
```

	year	birth	mortality
11	2011	9.2	9.7
12	2012	8.5	10.2
13	2013	7.9	10.2
14	2014	7.9	10.1
15	2015	8.3	10.5
16	2016	8.4	10.7
17	2017	8.4	10.7
18	2018	8.5	11.0

1.4.2 Alterar Valores

As duas formas mais simples de alterar valores na tabela é usando o método `at()` (baseado na etiqueta) ou `iat()` (baseado na posição). Os dois exemplos seguinte são equivalentes: alteram a taxa bruta da natalidade da primeira linha da tabela.

```
população.at[ 0, 'Natalidade' ] = 9.2
```

```
população.iat[ 0, 1 ] = 9.2
```

```
população.iloc[0, 1]
```

9.2

1.4.3 Acrescentar uma coluna

Podemos aplicar algumas expressões, por exemplo matemáticas, ao nossos dados e criar novas colunas a partir daí. O exemplo seguinte mostra a criação da coluna 'Diferença' que resulta da diferenças entre as duas taxas representadas na tabela.


```
população['Diferença'] = população.Natalidade - população.Mortalidade
população
```

	Ano	Natalidade	Mortalidade	Diferença
0	2011	9.2	9.7	-0.5
1	2012	8.5	10.2	-1.7
2	2013	7.9	10.2	-2.3
3	2014	7.9	10.1	-2.2
4	2015	8.3	10.5	-2.2
5	2016	8.4	10.7	-2.3
6	2017	8.4	10.7	-2.3
7	2018	8.5	11.0	-2.5

1.4.4 Rearranjar

A tabela população que temos estado a usar tem 8 linhas e 3 colunas: Ano, Natalidade e Mortalidade.

Podemos criar uma nova tabela trocando as linhas por colunas. Usando o método `transpose()` todas as colunas viram linhas, como no exemplo seguinte.

```
p1 = população.transpose()
p1
```

	0	1	2	3	4	5	6	7
Ano	2011.0	2012.0	2013.0	2014.0	2015.0	2016.0	2017.0	2018.0
Natalidade	9.2	8.5	7.9	7.9	8.3	8.4	8.4	8.5
Mortalidade	9.7	10.2	10.2	10.1	10.5	10.7	10.7	11.0

Como se vê, passamos a ter 8 colunas, indexadas de 0 a 7. Nalgumas situações, será útil que uma das colunas passe a ser o índice das colunas. Para tal, usa-se o método `set_index()` antes de `transpose()`, como se faz no exemplo seguinte:

```
pop = população.set_index('Ano').transpose()
pop
```

	Ano	2011	2012	2013	2014	2015	2016	2017	2018
Natalidade		9.2	8.5	7.9	7.9	8.3	8.4	8.4	8.5
Mortalidade		9.7	10.2	10.2	10.1	10.5	10.7	10.7	11.0

Como a coluna Ano era do tipo `int`, os índices das colunas são também do tipo `int`.

1.5 Leitura de ficheiros de dados

O Pandas apresenta um conjunto de funções para a leitura de ficheiros, todas começadas por `read_`, seguido da extensão do arquivo, e com o caminho do ficheiro como argumento obrigatório.

Por exemplo, a função `read_csv` consegue ler ficheiros do tipo `.csv`. Por definição, CSV é um formato de ficheiro que significa **c**omma-**s**eparated-**v**alues (valores separados por vírgulas). Isto significa que os campos de dados indicados neste formato normalmente são separados ou delimitados por uma vírgula. Assim, a leitura de um ficheiro csv com o pandas, seria feita da seguinte forma:

```
df = pd.read_csv('situacao_epidemiologica.csv')
```

Importa referir que o pandas lê sem problemas um ficheiro remoto, se lhe passarmos um endereço válido.

Após esta leitura do ficheiro, obtemos então uma das principais estruturas Pandas, um objeto `DataFrame`, neste caso chamado `df`.

De seguida, podemos observar a utilização do pandas sobre esta tabela, onde na mesma linha estamos a usar:

- um método para ordenar `data_relatorio` por ordem decrescente;
- a restringir apenas a duas colunas específicas;
- a aproveitar apenas as 10 primeiras linhas.

```
pandemia.sort_values(by=['data_relatorio'], ascending=False)[['data_relatorio', 'confirmados']].head(10)
```

	data_relatorio	confirmados
64	2020-05-06	26182
63	2020-05-05	25702
14	2020-05-03	25282
46	2020-05-02	25190
45	2020-05-01	25351
43	2020-04-30	25056
42	2020-04-29	24322
38	2020-04-28	24322
15	2020-04-27	24027
11	2020-04-26	23864

2 Exercícios

Com esta ficha de trabalho pretende-se apresentar alguns problemas exemplificativos da utilização do pandas.

1. Utilizando o ficheiro `situacao_epidemiologica.csv` previamente importado, escreva um código que retorne:
 - (a) o primeiro dia dos registos;
 - (b) em que dia se atingiram os mil óbitos (os óbitos estão registados cumulativamente. Isto é, para cada dia, estão indicados todos os óbitos registados até esse dia e não apenas os óbitos desse dia);
2. Usando o Pandas para ler o ficheiro `iris.csv` (atenção pois este ficheiro inclui o títulos das colunas), escreva um programa que responda às seguintes questões:

- (a) Quantas entradas de dados existem neste conjunto de dados?
 - (b) Qual o tipo de dados de cada coluna?
 - (c) Quais são os nomes das colunas?
 - (d) A coluna **species** corresponde aos nomes das espécies das flores. Estão também presentes quatro medidas básicas que podemos fazer de uma flor: o comprimento e largura das pétalas e o comprimento e largura do caule. Quantas espécies de flores estão incluídas nos dados?
 - (e) Os dados fornecidos contêm um erro na linha 35 e na linha 38. A linha 35 deveria ser 4.9,3.1,1.5,0.2,"setosa", e a linha 38 deveria ser 4.9,3.6,1.4,0.1,"setosa". Leia estas linhas, veja o estado atual das mesmas e depois proceda às correções nas posições que estão erradas no DataFrame.
 - (f) Adicione duas colunas, **Petal.Ratio** (rácio entre a **petal length** e a **petal width**) e **Sepal.Ratio** (rácio entre a **sepal length** e a **sepal width**).
3. A limpeza de dados após a recolha dos mesmos é um passo muito importante para garantir a fiabilidade dos programas desenvolvidos. Usando o Pandas para ler o ficheiro `meteoritos.csv`, proceda à limpeza destes dados, garantindo que inclui os seguintes pontos:
- a coluna `year` deve ser do tipo inteiro (`Int64Dtype`) mas está como float. Faça a conversão.
 - visto que os dados foram recolhidos até 2016, verifique se existe alguma linha com ano superior e proceda à sua eliminação. Elimine também as linhas com a massa igual a zero e as linhas com a geolocalização igual a (0.000000, 0.000000). Dica: use o método `drop()`.
 - remova as linhas que tenham valores em falta (nulos). Dica: use o método `dropna()`
- Por fim, exporte os dados do DataFrame para um ficheiro csv chamado `meteoritos_limpo.csv`. Dica: use o método `to_csv()`.
4. Usando o Pandas para ler o ficheiro `meteoritos_limpo.csv` criando no exercício anterior, escreva um conjunto de instruções que responda às seguintes questões:
- (a) Qual foi o último meteorito observado a cair?
 - (b) Qual foi o último meteorito encontrado?
 - (c) Qual o número de meteoritos registados por tipo de queda (observado vs encontrado)? Dica: use o método `value_counts()`
 - (d) Qual o ano com mais meteoritos encontrados? Dica: use o método `idxmax()` junto com o `value_counts()`
 - (e) Quais os 5 meteoritos mais pesados?
 - (f) Qual o meteorito mais pesado de cada ano? Dica: use os métodos `group_by` e `agr`.
5. Usando o Pandas para ler o ficheiro `poluicao.csv` (inclua `encoding="utf-8"` na importação do ficheiro), escreva um conjunto de instruções que efetue as seguintes operações:
- (a) Criar duas novas colunas (`WaterQuality` e `AirPollution`) a partir das que já existem usando o método `apply()`.
 - (b) Indicar as cidades com poluição de ar e água acima das médias.

- (c) Indicar o país mais afetado a nível de poluição de água e o país mais afetado a nível de poluição de ar.
- (d) Criar uma nova coluna com a média da qualidade no geral (ar e água).
- (e) Indicar a(s) cidade(s) dos Estados Unidos mais poluída(s).