

Programação Python

Aula 09: DataFrames (pandas)

Prof. Eduardo Corrêa Gonçalves

29/03/2021

Sumário

Introdução

O que é DataFrame?

Criação

Propriedades e Operações Básicas

Operações de Data Wrangling

Introdução (1/4)

- O que é DataFrame?
 - ED utilizada para representar dados tabulares, estruturados em **linhas** e **colunas**, em memória.

	<i>nome</i>	<i>continente</i>	<i>extensao</i>
AR	Argentina	América	2780
BR	Brasil	América	8511
FR	França	Europa	644
IT	Itália	Europa	301
UK	Reino Unido	Europa	244

df_paises

- Características:
 - **Indispensável** – esta é a ED mais importante da Ciência de Dados.
 - **Eficiente** – internamente, faz uso de arrays NumPy.
 - Ideal para a **estruturação, limpeza, transformação e análise** de bases de dados.

Introdução (2/4)

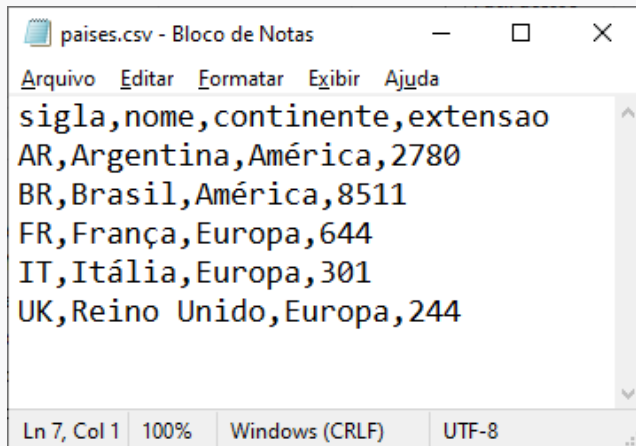
- O que é DataFrame?
 - Cada coluna é, na verdade, uma Series.
 - O DataFrame é um **dicionário de Series**, todas do mesmo tamanho (*size*).
 - Colunas possuem **rótulos**.
 - Opcionalmente, as linhas também podem ser rotuladas.
- No DataFrame `df_paises` as linhas são indexadas pela “sigla do país”.
- Colunas:
 - “nome” (nome do país);
 - “continente” (nome do continente)
 - “extensão” (extensão territorial em 1000 km²).

`df_paises`

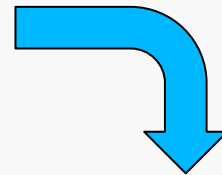
	<i>nome</i>	<i>continente</i>	<i>extensao</i>
AR	Argentina	América	2780
BR	Brasil	América	8511
FR	França	Europa	644
IT	Itália	Europa	301
UK	Reino Unido	Europa	244

Introdução (3/4)

- Por que o DataFrame é tão importante?
 - Porque é a estrutura mais adequada para estruturar bases de dados em memória.
 - Uma base no formato tabular (ex.: CSV) é mapeada diretamente para um Data Frame.



```
países.csv - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
sigla,nome,continente,extensao
AR,Argentina,América,2780
BR,Brasil,América,8511
FR,França,Europa,644
IT,Itália,Europa,301
UK,Reino Unido,Europa,244
Ln 7, Col 1 100% Windows (CRLF) UTF-8
```



df_paises

	<i>nome</i>	<i>continente</i>	<i>extensao</i>
AR	Argentina	América	2780
BR	Brasil	América	8511
FR	França	Europa	644
IT	Itália	Europa	301
UK	Reino Unido	Europa	244

Introdução (4/4)

- **Importação da Biblioteca pandas**

- Assim como a Series, a ED DataFrame **não** faz parte do Python padrão, sendo oferecida pela biblioteca 'pandas'.
 - Lembrando que a 'pandas' está disponível no Google Colab, WinPython, Anaconda e qualquer distribuição Python voltada para Ciência de Dados.
 - Mas não vem junto como CPython (precisa ser instalada a parte)
- Para utilizar a 'pandas', você precisa usar o comando **import** (além de importar a biblioteca, a renomeamos para "pd"):

import pandas as pd.

- *Nem sempre vou mostrar esse import nos slides para poupar espaço.*
- *Mas você precisa inseri-lo em seus programas.*

Criação de DataFrames (1/2)

- Criando DataFrames

- Utiliza-se o construtor `pd.DataFrame()`
- Basta especificar um dicionário com os dados.
- Opcionalmente, podemos usar o parâmetro `index` para passar os índices das linhas.

```
import pandas as pd

#cria o DataFrame
dados = {'nome': ['Argentina', 'Brasil', 'França', 'Itália',
                 'Reino Unido'],
         'continente': ['América', 'América', 'Europa', 'Europa',
                       'Europa'],
         'extensao': [2780, 8511, 644, 301, 244]}

siglas = ['AR', 'BR', 'FR', 'IT', 'UK']
df_paises = pd.DataFrame(dados, index=siglas)

#imprime o DataFrame
print(df_paises)
```

Criação de DataFrame (2/2)

- Mas na prática...
 - Porém, na maioria das situações práticas os data frames serão carregados a partir de arquivos.

```
df_paises = pd.read_csv("https://raw.githubusercontent.com/edubd/bd/main/paises.csv", index_col = "sigla")  
  
print(df_paises)
```

```
>>>
```

	nome	continente	extensao
sigla			
AR	Argentina	América	2780
BR	Brasil	América	8511
FR	França	Europa	644
IT	Itália	Europa	301
UK	Reino Unido	Europa	244

Propriedades dos DataFrames

```
#propriedades do DataFrame
print('número de linhas: ', df_paises.shape[0])
print('número de colunas: ', df_paises.shape[1])
print('rótulos das linhas: ', df_paises.index)
print('rótulos das colunas: ', df_paises.columns)
print('tipo (type): ', type(df_paises))
print('dtypes das colunas:\n', df_paises.dtypes)
print('dtype dos rótulos das linhas:',
      df_paises.index.dtype)
```

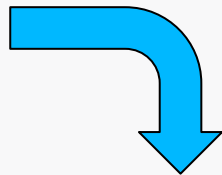
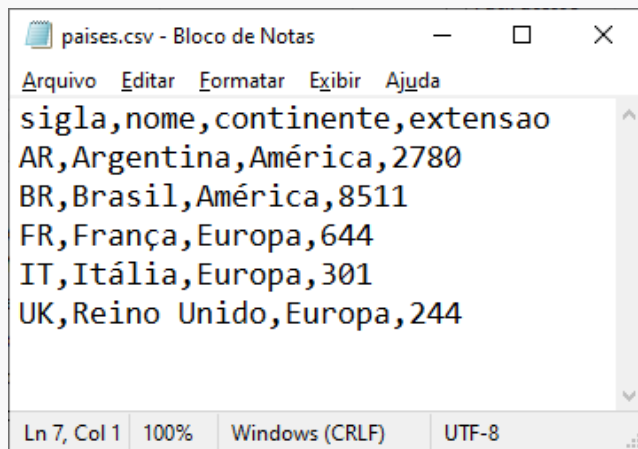
```
número de linhas: 5
número de colunas: 3
rótulos das linhas: Index(['AR', 'BR', 'FR', 'IT', 'UK'],
dtype='object', name='sigla')
rótulos das colunas: Index(['nome', 'continente',
'extensao'], dtype='object')
tipo (type): <class 'pandas.core.frame.DataFrame'>
dtypes das colunas:
  nome          object
continente      object
extensao        int64
dtype: object
dtype dos rótulos das linhas: object
```

Operações

- **Operações Básicas**
 - Importação de BD para DataFrame.
 - Indexação e Fatiamento
 - Inserindo e Excluindo Linhas

Importação de BD para DataFrame (1/2)

- **Exemplo 1**: arquivo CSV separado por vírgula
 - Se os dados do arquivo estão separados por vírgula, basta indicar o caminho do arquivo.
 - O parâmetro “index_col” pode ser usado para permitir com que uma coluna seja usada como índice das linhas (nesse caso, “sigla”)



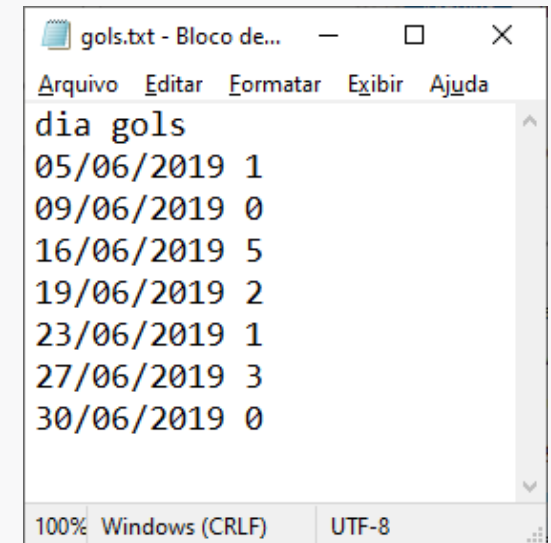
```
nome_arq = "https://raw.githubusercontent.com/edubd/bd/  
main/paises.csv"
```

```
df_paises = pd.read_csv(nome_arq, index_col = "sigla")
```


Importação de BD para Series

- Exemplo 3: Importando arquivo para Series

- Caso nosso arquivo tenha apenas 1 ou 2 colunas, podemos importá-lo para uma Series em vez de um DataFrame.
 - Basta usar o parâmetro **squeeze**.
 - Nesse exemplo, BD tem os gols marcados por um time em vários jogos.
 - Importamos fazendo:
 - Vetor de índices = dia
 - Vetor de dados = gols



```
nome_arq = "https://raw.githubusercontent.com/edubd/bd/main/gols.txt"

s_gols = pd.read_csv(nome_arq, sep=" ",
                     squeeze=True, index_col=0)
#converte o tipo do índice para datetime
serie_gols.index = pd.to_datetime(serie_gols.index,
                                  format='%d/%m/%Y')
```

Indexação Básica (1/2)

- **Métodos:**

- **`iloc()`, `iat()`** :
 - Para indexação baseada na posição da linha (número inteiro).
- **`loc()`, `at()`**
 - Para acessar linhas através de seus rótulos

Sintaxe	Explicação
<code>d.iloc[i][j]</code>	retorna o valor da célula que ocupa a linha i, coluna j
<code>d.iat[i,j]</code>	retorna o valor da célula que ocupa a linha i, coluna j
<code>d.iloc[i]['col']</code>	retorna o valor da célula que ocupa a linha i, coluna denominada 'col'
<code>d.loc['idx'][j]</code>	retorna o valor da célula que ocupa a linha do índice de rótulo 'idx', coluna j
<code>d.loc['idx']['col']</code>	retorna o valor da célula que ocupa a linha do índice de rótulo 'idx', coluna denominada 'col'
<code>d.at['idx','col']</code>	retorna o valor da célula que ocupa a linha do índice de rótulo 'idx', coluna denominada 'col'

Indexação (2/2)

- Exemplo:

		[0]	[1]	[2]
		<i>nome</i>	<i>continente</i>	<i>extensao</i>
[0]	AR	Argentina	América	2780
[1]	BR	Brasil	América	8511
[2]	FR	França	Europa	644
[3]	IT	Itália	Europa	301
[4]	UK	Reino Unido	Europa	244

df_paises

Exemplo	Resultado
<code>df_paises.iloc[1][0]</code>	Brasil
<code>df_paises.iat[1,0]</code>	Brasil
<code>df_paises.iloc[3]['extensao']</code>	301
<code>df_paises.loc['FR'][1]</code>	Europa
<code>df_paises.loc['FR']['nome']</code>	França

Fatiamento Básico (1/3)

- **Métodos:**

- Na tabela abaixo, a forma de recuperar uma coluna ou linha inteira em uma Series

Sintaxe	Explicação
d.['col']	retorna a coluna de nome 'col' (toda a coluna)
d.col	idem
d.loc['idx']	retorna a linha associada ao rótulo 'idx' (linha inteira)
d.iloc[i]	Retorna a linha que ocupa a posição i (linha inteira)

- Com os métodos **iloc()** e **loc()** podemos fazer outros tipos de fatiamentos, para gerar “sub-DataFrames”.
 - Neste caso, basta aplicar a sintaxe idêntica a usada na NumPy.
 - Lembrando que as fatias são visões do DataFrame original.

Fatiamento Básico (2/3)

	<i>nome</i>	<i>continente</i>	<i>extensão</i>
AR	Argentina	América	2780
BR	Brasil	América	8511
FR	França	Europa	644
IT	Itália	Europa	301
UK	Reino Unido	Europa	244

`df_paises.nome`

- Fatia toda a coluna nome

	<i>nome</i>	<i>continente</i>	<i>extensão</i>
AR	Argentina	América	2780
BR	Brasil	América	8511
FR	França	Europa	644
IT	Itália	Europa	301
UK	Reino Unido	Europa	244

`df_paises.iloc[:3, :2]`

- Fatia com as linhas 0 a 3, colunas 0 e 1

Fatiamento Básico (3/3)

	<i>nome</i>	<i>continente</i>	<i>extensão</i>
AR	Argentina	América	2780
BR	Brasil	América	8511
FR	França	Europa	644
IT	Itália	Europa	301
UK	Reino Unido	Europa	244

`df_países.iloc[2]`

- Fatia toda a coluna nome

	<i>nome</i>	<i>continente</i>	<i>extensão</i>
AR	Argentina	América	2780
BR	Brasil	América	8511
FR	França	Europa	644
IT	Itália	Europa	301
UK	Reino Unido	Europa	244

`df_países.iloc[-2:, 1:3]`

- Fatia com as duas últimas linhas, colunas 1 e 2.

Modificação Básica

```
#8-modificação básica
```

```
 #(a) insere o país Japão
```

```
df_países.loc['JP'] = {'nome': 'Japão',  
                       'continente': 'Ásia',  
                       'extensao': 372}
```

```
print(df_países)
```

```
 #(b) altera a extensão do Brasil
```

```
df_países.at['BR', 'extensao'] = 8512
```

```
 #(c) remove o Japão
```

```
df_países = df_países.drop(['JP'])
```

```
print(df_países)
```

Operações

- Operações de **Data Wrangling**:
 - Projeção
 - Seleção
 - Modificação
 - Estatísticas Descritivas e Gráficos
 - Discretização
 - Limpeza
 - Combinando DataFrames (junção)

Data Wrangling (1/17)

- Base de Dados

- Nos exemplo, utilizaremos o BD “**lojas.csv**” (13 linhas x 6 colunas).

```
df_lojas = pd.read_csv("https://raw.githubusercontent.com/edubd/bd/main/lojas.csv")
```

raiz	sufixo	fantasia	po	salario	uf
22222222	0001	PYTHONSHOP I	20	35000	SP
22222222	0002	PYTHONSHOP II	5	4000	RJ
22222222	0003	PYTHONSHOP III	15	31000	MG
33333333	0001	NPSEV LOJA A	8	9200	MG
33333333	0002	NPSEV LOJA B	0	0	MG
44444444	0001	XYZ MATRIZ	112	250000	RJ
44444444	0003	XYZ CENTRO	101	103900	SP
44444444	0004	XYZ NORTE	48	60000	RJ
44444444	0005	XYZ SUL	50	65000	RJ
44444444	0010	XYZ LITORAL	1	2500	RJ
55555555	0001	JAVASERV	40	40000	SP
66666666	0001	CAFEDB	12	12900	RS
99999999	0001	PANDAS KING	1	3000	MG

- raiz:** código que identifica uma empresa.
- sufixo:** código de 4 dígitos que, junto com a raiz, identifica uma loja da empresa.
- fantasia:** nome fantasia da loja.
- po:** pessoal ocupado (quantidade de funcionários da loja).
- salario:** valor mensal gasto pela loja com salários
- uf:** uf onde localiza-se a loja

Data Wrangling (2/17)

- **Projeção:** gera um novo DataFrame a partir da extração de algumas colunas de um outro.
 - Útil para **descartar atributos** que não precisarão ser envolvidas em um estudo estatístico.

```
#gera um novo DataFrame apenas com a uf, po e salário  
df_uf_po_sal = df_lojas[['uf', 'po', 'salario']]  
print(df_uf_po_sal)
```

	uf	po	salario
0	SP	20	35000
1	RJ	5	4000
2	MG	15	31000
3	MG	8	9200
4	MG	0	0
...			
11	RS	12	12900
12	MG	1	3000

Data Wrangling (3/17)

- **Seleção:** gera um novo DataFrame a partir da extração de algumas linhas de interesse (filtragem de linhas)
 - Implementada com o uso de **indexação booleana**.

```
#EXEMPLO 1: seleciona apenas as lojas do RJ  
  
v = (df_lojas['uf']=='RJ') #gera a Series booleana  
df_rj = df_lojas[v]       #filtra as linhas
```

	raiz	sufixo	fantasia	po	salario	uf
1	22222222	2	PYTHONSHOP II	5	4000	RJ
5	44444444	1	XYZ MATRIZ	112	250000	RJ
7	44444444	4	XYZ NORTE	48	60000	RJ
8	44444444	5	XYZ SUL	50	65000	RJ
9	44444444	10	XYZ LITORAL	1	2500	RJ

Data Wrangling (4/17)

- **Seleção:** gera um novo DataFrame a partir da extração de algumas linhas de interesse (filtragem de linhas)
 - Implementada com o uso de **indexação booleana**.

```
#EXEMPLO 2: seleciona apenas as lojas do RJ
#           com 50 ou mais funcionários

#gera a Series booleana
v = (df_lojas['uf']=='RJ') & (df_lojas['po']>=50)

#filtra as linhas
df_rj_grandes = df_lojas[v]
```

	raiz	sufixo	fantasia	po	salario	uf
5	44444444	1	XYZ MATRIZ	112	250000	RJ
8	44444444	5	XYZ SUL	50	65000	RJ

Data Wrangling (5/17)

- **Estatísticas Básicas:**
 - Ex1.: estatísticas da variável salário (contínua)

```
#Data Wrangling 3: Estatísticas Descritivas
print('- Estatísticas da variável salário: ')
print('média          : {:.2f}'.format(df_lojas['salario'].mean()))
print('mediana         : {:.2f}'.format(df_lojas['salario'].median()))
print('variância        : {:.2f}'.format(df_lojas['salario'].var()))
print('desv. padrão: {:.2f}'.format(df_lojas['salario'].std()))
```

```
- Estatísticas da variável salário:
média          : 47423.08
mediana         : 31000.00
variância        : 4662681923.08
desvio padrão: 68283.83
```

Data Wrangling (6/17)

- **Estatísticas Básicas:**
 - Ex2.: estatísticas da variável uf (categórica)

```
#Data Wrangling 3: Estatísticas Descritivas
print('\n- Estatísticas da variável uf: ')
print('moda: ', df_lojas['uf'].mode())
#retorna todas as categorias distintas
print('domínio: ', df_lojas['uf'].unique())
print('freq. das categorias:')
#retorna a freq. de cada categoria
print(df_lojas['uf'].value_counts())
```

```
- Estatísticas da variável uf:
moda: 0      RJ
dtype: object
domínio: ['SP' 'RJ' 'MG' 'RS']
freq. das categorias:
RJ      5
MG      4
SP      3
RS      1
Name: uf, dtype: int64
```

Data Wrangling (7/17)

- **Estatísticas Básicas:**
 - Ex3.: tabulação – soma do PO por UF

```
#produzindo uma tabulação
#(i)-Gera variável "grouped" onde a chave é "uf" e a medida "po"
grupo_po_uf = df_lojas['po'].groupby(df_lojas['uf'])

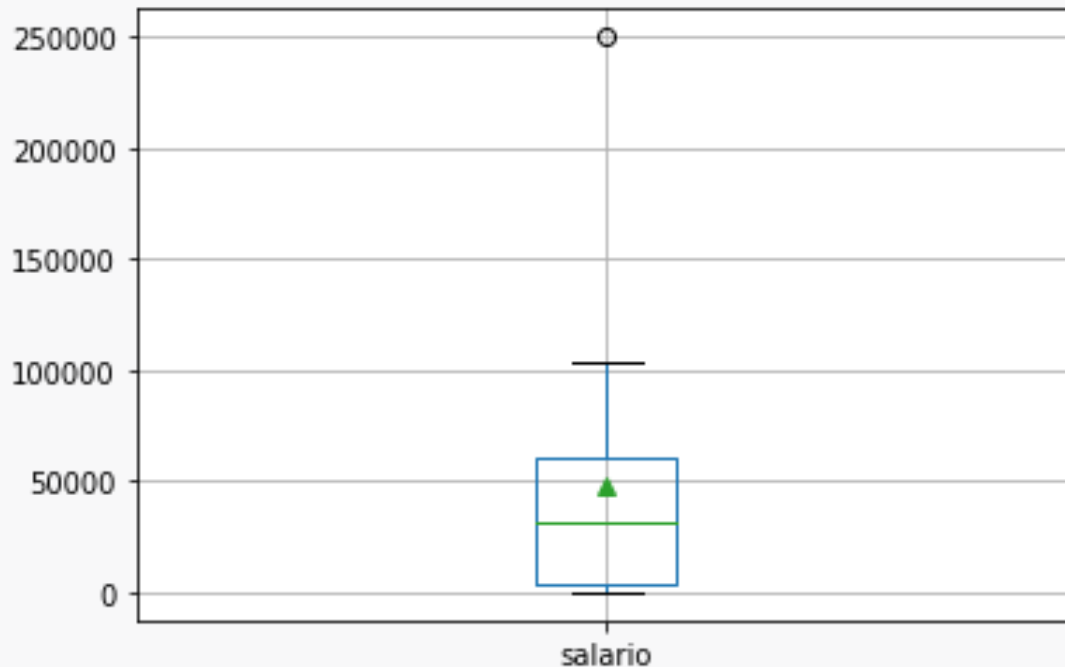
#(ii)-Computa agregados a partir da variável gerada
print('- Soma do PO, por UF: ', grupo_po_uf.sum())
```

```
- Soma do PO, por UF:  uf
MG          24
RJ         216
RS          12
SP         161
Name: po, dtype: int64
```

Data Wrangling (8/17)

- Gráficos
 - Ex1.: gera boxplot da variável salário.

```
#Gera o boxplot  
boxplot = df_lojas.boxplot(column=['salario'], showmeans=True)
```

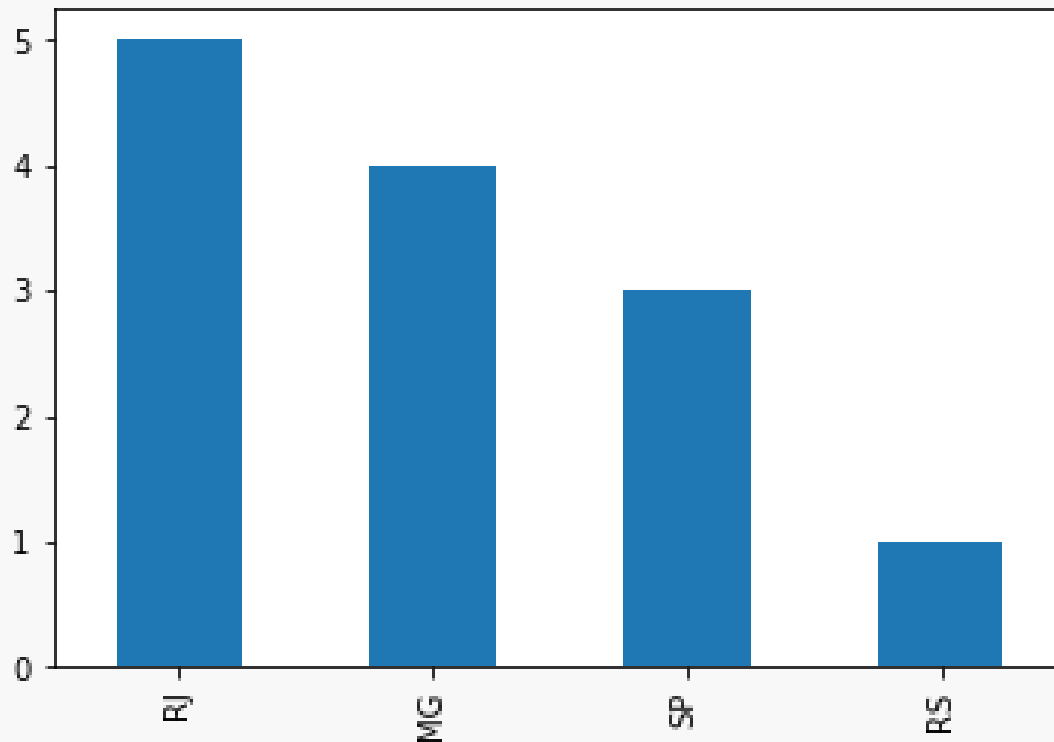


Data Wrangling (9/17)

- Gráficos

- Ex2.: gráfico de barras com total de lojas por UF

```
tot_lojas_uf = df_lojas['uf'].value_counts()  
barras = tot_lojas_uf.plot(kind="bar", legend=False)
```



Data Wrangling (10/17)

- **Discretização:** Discretizar um atributo contínuo significa converter os seus valores para um conjunto reduzido de valores discretos ou categóricos.
 - O próximo exemplo mostra como converter o atributo “po” para “faixa_po” a partir da **definição de uma função** e de sua aplicação com o emprego do método **apply()**.

```
#Data Wrangling 4: Discretização

#(1)-Define uma função para discretizar o PO
def discretiza_po(po):
    if po == 0: return "0";
    elif po <= 10: return "1-10";
    elif po <= 50: return "11-50";
    elif po > 50: return ">50"
    else: return "desconhecido";

#(2)-Cria a coluna "fx_po" usando apply (faixa de PO)
df_lojas['faixa_po'] = df_lojas['po'].apply(discretiza_po)

#(3)-Podemos remover definitivamente o 'po' se quisermos
df_lojas = df_lojas.drop(columns=['po'])
```

Data Wrangling (11/17)

- Discretização

```
print(df_lojas)
```

	raiz	sufixo	fantasia	salario	uf	faixa_po
0	22222222	1	PYTHONSHOP I	35000	SP	11-50
1	22222222	2	PYTHONSHOP II	4000	RJ	1-10
2	22222222	3	PYTHONSHOP III	31000	MG	11-50
3	33333333	1	NPSEV LOJA A	9200	MG	1-10
4	33333333	2	NPSEV LOJA B	0	MG	0
5	44444444	1	XYZ MATRIZ	250000	RJ	>50
6	44444444	3	XYZ CENTRO	103900	SP	>50
7	44444444	4	XYZ NORTE	60000	RJ	11-50
8	44444444	5	XYZ SUL	65000	RJ	11-50
9	44444444	10	XYZ LITORAL	2500	RJ	1-10
10	55555555	1	JAVASERV	40000	SP	11-50
11	66666666	1	CAFEDB	12900	RS	11-50
12	99999999	1	PANDAS KING	3000	MG	1-10

Data Wrangling (12/17)

- **Limpeza:** atividade de data wrangling que consiste em eliminar sujeira e informações irrelevantes de uma base de dados.
 - No próximo exemplo, o método de string **rstrip()** é aplicado para remover os espaços em branco à direita do atributo “fantasia”.

```
#Data Wrangling 5: Limpeza

#(1)-Utiliza o apply para remover os espaços em branco
df_lojas['fantasia'] = df_lojas['fantasia'].apply(str.rstrip)
print(df_lojas)
```

	raiz	sufixo	fantasia	salario	uf	faixa_po
0	22222222	1	PYTHONSHOP I	35000	SP	11-50
1	22222222	2	PYTHONSHOP II	4000	RJ	1-10
2	22222222	3	PYTHONSHOP III	31000	MG	11-50
3	33333333	1	NPSEV LOJA A	9200	MG	1-10
4	33333333	2	NPSEV LOJA B	0	MG	0
...						
11	66666666	1	CAFEDB	12900	RS	11-50
12	99999999	1	PANDAS KING	3000	MG	1-10

Data Wrangling (13/17)

- **Combinando DataFrames:** vamos combinar as linhas do DataFrame “df_lojas” com as linhas de outro DataFrame chamado “df_emp”
 - Isso é feito através da variável em comum “raiz”.
 - A operação é conhecida como **junção** ou **merge**.

```
df_emp = pd.read_csv("https://raw.githubusercontent.com/edubd/bd/main/emp.csv")
```

raiz	razao	natjur
11111111	UNIVERSIDADE ALPHA	1
22222222	PYTHONISTA	2
33333333	NPSERV	1
44444444	MERCADO XYZ	2
55555555	JAVANESE BRASIL	2
66666666	CAFEDB	2
88888888	ABG	1

- **raiz:** código que identifica uma empresa.
- **razao:** razão social da empresa.
- **natjur:** natureza jurídica da empresa (1=empresa pública, 2=empresa privada).

Data Wrangling (14/17)

- Combinando DataFrames

```
#Data Wrangling 6: Junção
```

```
#(1) INNER JOIN
```

```
c1 = pd.merge(df_emps, df_lojas, how='inner', on='raiz')  
print('resultado do INNER JOIN: ')  
print(c1)  
print('-----')
```

```
#(2) LEFT JOIN
```

```
c2 = pd.merge(df_emps, df_lojas, how='left', on='raiz')  
print('resultado do LEFT JOIN: ')  
print(c2)  
print('-----')
```

```
#(3) FULL OUTER JOIN
```

```
c3 = pd.merge(df_emps, df_lojas, how='outer', on='raiz')  
print('resultado do FULL OUTER JOIN: ')  
print(c3)
```

Data Wrangling (15/17)

- Combinando DataFrames – INNER JOIN

resultado do INNER JOIN:

	raiz	razao	natjur	...	salario	uf	faixa_po
0	22222222	PYTHONISTA	2	...	35000	SP	11-50
1	22222222	PYTHONISTA	2	...	4000	RJ	1-10
2	22222222	PYTHONISTA	2	...	31000	MG	11-50
3	33333333	NPSEV	1	...	9200	MG	1-10
4	33333333	NPSEV	1	...	0	MG	0
5	44444444	MERCADO XYZ	2	...	250000	RJ	>50
6	44444444	MERCADO XYZ	2	...	103900	SP	>50
7	44444444	MERCADO XYZ	2	...	60000	RJ	11-50
8	44444444	MERCADO XYZ	2	...	65000	RJ	11-50
9	44444444	MERCADO XYZ	2	...	2500	RJ	1-10
10	55555555	JAVANESE BRASIL	2	...	40000	SP	11-50
11	66666666	CAFEDB	2	...	12900	RS	11-50

Data Wrangling (16/17)

- Combinando DataFrames – LEFT JOIN

resultado do LEFT JOIN:

	raiz	razao	natjur	...	salario	uf	faixa_po
0	11111111	UNIVERSIDADE	ALPHA	1	...	NaN	NaN
1	22222222	PYTHONISTA		2	...	35000.0	SP
2	22222222	PYTHONISTA		2	...	4000.0	RJ
3	22222222	PYTHONISTA		2	...	31000.0	MG
4	33333333	NPSERV		1	...	9200.0	MG
5	33333333	NPSERV		1	...	0.0	MG
6	44444444	MERCADO	XYZ	2	...	250000.0	RJ
7	44444444	MERCADO	XYZ	2	...	103900.0	SP
8	44444444	MERCADO	XYZ	2	...	60000.0	RJ
9	44444444	MERCADO	XYZ	2	...	65000.0	RJ
10	44444444	MERCADO	XYZ	2	...	2500.0	RJ
11	55555555	JAVANESE	BRASIL	2	...	40000.0	SP
12	66666666	CAFEDB		2	...	12900.0	RS
13	88888888	ABG		1	...	NaN	NaN

Data Wrangling (17/17)

- Combinando DataFrames – FULL OUTER JOIN

resultado do FULL OUTER JOIN:

	raiz	razao	natjur	...	salario	uf	faixa_po
0	11111111	UNIVERSIDADE	ALPHA	1.0	...	NaN	NaN
1	22222222	PYTHONISTA		2.0	...	35000.0	SP
2	22222222	PYTHONISTA		2.0	...	4000.0	RJ
3	22222222	PYTHONISTA		2.0	...	31000.0	MG
4	33333333	NPSERV		1.0	...	9200.0	MG
5	33333333	NPSERV		1.0	...	0.0	MG
6	44444444	MERCADO	XYZ	2.0	...	250000.0	RJ
7	44444444	MERCADO	XYZ	2.0	...	103900.0	SP
8	44444444	MERCADO	XYZ	2.0	...	60000.0	RJ
9	44444444	MERCADO	XYZ	2.0	...	65000.0	RJ
10	44444444	MERCADO	XYZ	2.0	...	2500.0	RJ
11	55555555	JAVANESE	BRASIL	2.0	...	40000.0	SP
12	66666666	CAFEDB		2.0	...	12900.0	RS
13	88888888	ABG		1.0	...	NaN	NaN
14	99999999	NaN	NaN	NaN	...	3000.0	MG

Tarefa

- DOJO: Data Wrangling da base “flags.csv”

Referências

- Corrêa, E. (2020). “Meu Primeiro Livro de Python”. V 2.0.0, edubd, 2020. (*capítulo 7*).
 - Disponível em: https://github.com/edubd/meu_primeiro_livro_de_python
- Corrêa, E. (2020). “Pandas Python: Data Wrangling para Ciência de Dados”. Casa do Código.
- Pandas (2021). <https://pandas.pydata.org>