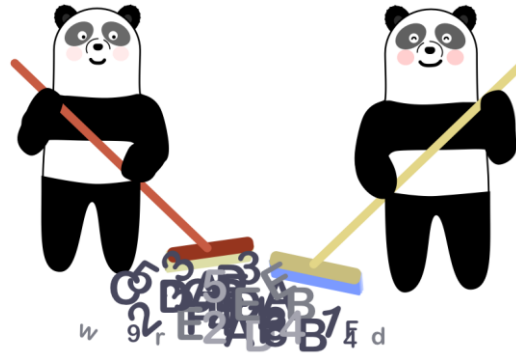


Engenharia de Dados – Parte 02

Wellington Franco
Universidade Federal do Ceará – UFC
Campus da UFC em Crateús
wellington@crateus.ufc.br



Manipulação e Limpeza de Dados

Manipulando os Dados

- Como já mencionamos anteriormente, iremos trabalhar utilizando *DataFrames* por serem mais organizados na hora de visualizar, manipular e limpar dados.
- A biblioteca responsável por gerar *DataFrames* em *Python* é a Pandas

```
In [1]: import pandas as pd
```



Primeiros passos

Ao declararmos um *DataFrame*, podemos fazer de duas maneiras:

1) De forma **default**, no qual apenas declaramos os dados dentro da função `pd.DataFrame`:

```
In [2]: # Dataframe default  
matrix_default_df = pd.DataFrame(data=[[1,2,3], [4,5,6], [7,8,9]])
```

```
In [3]: matrix_default_df
```

Out[3]:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Primeiros passos

2) Inserindo os dados e nomeando as linhas (index) e colunas (columns)

```
In [5]: # DataFrame personalizado  
matrix_df = pd.DataFrame(data=[[1,2,3],[4,5,6],[7,8,9]], index=(0,'A','$'), columns=['colA', 'colB', 'colC'])
```

```
In [6]: matrix_df
```

Out[6]:

	colA	colB	colC
0	1	2	3
A	4	5	6
\$	7	8	9

Acessando Linhas e Colunas no DataFrame

- **Linhas:**

Podemos fazer manipulação na linha de duas formas: pelo *index original* ou pelos rótulos dados. Para isso, utilizamos:

- `loc[]`: no qual retorna a linha de acordo com o rótulo dado
- `iloc[]`: faz a busca de acordo com o *index original* do DataFrame

	colA	colB	colC
0	0	1	2
1	A	4	5
2	\$	7	8

```
In [7]: print(matrix_df.loc['A'])
```

```
colA    4
colB    5
colC    6
Name: A, dtype: int64
```

```
In [8]: print(matrix_df.iloc[2])
```

```
colA    7
colB    8
colC    9
Name: $, dtype: int64
```

Acessando Linhas e Colunas no DataFrame

- **Colunas:**

Podemos fazer manipulação nas colunas simplesmente das seguintes formas:

- Simplesmente digitando o nome da coluna entre colchetes;
- `iloc[]`: utilizando a função `iloc` e usando dois argumentos `[arg1, arg2]`:
 - `arg1`) dois-pontos: para trazer todas as linhas
 - `arg2`) o *index* da coluna

```
In [9]: matrix_df['colA']
```

```
Out[9]: 0    1  
A      4  
$      7  
Name: colA, dtype: int64
```

```
In [10]: matrix_df.iloc[:,0]
```

```
Out[10]: 0    1  
A      4  
$      7  
Name: colA, dtype: int64
```

Adicionando Linhas e Colunas no DataFrame

- Seja o seguinte DataFrame:

```
In [11]: # Seja o seguinte dataframe
df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), index=[2.5, 12.6, 4.8], columns=[48, 49, 50])
```

```
In [12]: df
```

Out[12]:

	48	49	50
2.5	1	2	3
12.6	4	5	6
4.8	7	8	9

OBS: Iremos utilizar rótulos “estranhos” para linhas e colunas no intuito de não confundir com os index originais de cada uma.

Adicionando Linhas e Colunas no DataFrame

- **Adicionando Linha:**

Se utilizar a função `loc`, ele gerará uma linha nova contendo os valores passados. O rótulo da linha será o valor passado como argumento:

Rótulo da linha

elementos da linha

```
In [47]: df.loc[3] = [11, 12, 13]  
df
```

Out[47]:

	48	49	50
2.5	1	2	3
12.6	4	5	6
4.8	60	50	40
3.0	11	12	13

É importante que a quantidade de elementos inseridos seja na mesma quantidade e na respectiva ordem para cada coluna!

Adicionando Linhas e Colunas no DataFrame

- **Adicionando Linha:**

Se utilizar a função `iloc`, ele apenas irá substituir os valores da linha do index passado como argumento

```
In [13]: df.iloc[2] = [60, 50, 40]  
df
```

Out[13]:

	48	49	50	
0	2.5	1	2	3
1	12.6	4	5	6
2	4.8	60	50	40


iloc[]

Como a função `iloc[]` serve para acessar o index original do DataFrame, se você tentar inserir `df.iloc[3]` com esses valores, ele não irá encontrar o index 3 e exibirá mensagem de erro!

Adicionando Linhas e Colunas no DataFrame

- **Adicionando Coluna:**

Para adicionar uma coluna, basta inserir o nome dentro do argumento e atribuir valores ao dataframe. Neste exemplo, adicionamos uma coluna D com os seus valores iguais aos do index do Dataframe

```
In [15]: df['D'] = df.index
```

```
df
```

```
Out[15]:
```

	48	49	50	D
2.5	1	2	3	2.5
12.6	4	5	6	12.6
4.8	60	50	40	4.8
3.0	11	12	13	3.0

Resetando o index do DataFrame

- Essa função serve para tornar os *index* do DataFrame com formato padrão:

```
In [16]: #Veja os index atuais do seu dataframe
df
```

Out[16]:

	48	49	50	D
2.5	1	2	3	2.5
12.6	4	5	6	12.6
4.8	60	50	40	4.8
3.0	11	12	13	3.0

Valores de index
que setamos
inicialmente

```
In [17]: # Use `reset_index()` para usar valores padrões
df_reset = df.reset_index(level=0, drop=True)
df_reset
```

Out[17]:

	48	49	50	D
0	1	2	3	2.5
1	4	5	6	12.6
2	60	50	40	4.8
3	11	12	13	3.0

Se `drop=False`, ele não descarta a coluna de index antiga, ele gera uma nova coluna no DataFrame e a insere

Removendo Linhas e Colunas do DataFrame

- Seja o seguinte DataFrame:

```
In [18]: df1 = pd.DataFrame(data=[[1,2,3],[4,5,6],[7,8,9]], index=(3,'AA','BB'), columns=['A', 'B', 'C'])  
df1
```

Out[18]:

	A	B	C
3	1	2	3
AA	4	5	6
BB	7	8	9

Removendo Linhas e Colunas do DataFrame

- **Linhas:** para remover linhas, é possível remover de duas formas:
 - Removendo pelo nome da linha
 - Removendo pelo seu index
- O argumento `axis` serve para indicar o que se deseja remover: 0 para linha e 1 para coluna. O *default* é zero!
- O argumento *inplace* serve para que a remoção seja efetuada sem precisar atribuir o resultado em uma nova variável. Por *default* ela é *False*, e serve apenas para “simular” como ficará o resultado após a remoção.

```
In [19]: df1.drop('AA', axis=0, inplace=True)  
df1
```

Out[19]:

	A	B	C
3	1	2	3
BB	7	8	9

```
In [20]: df1.drop(df1.index[1], axis=0, inplace=True)  
df1
```

Out[20]:

	A	B	C
3	1	2	3

Removendo Linhas e Colunas do DataFrame

- Seja o seguinte DataFrame:

```
In [21]: df2 = pd.DataFrame(data=[[1,2,3],[4,5,6],[7,8,9]], index=(3,'AA','BB'), columns=['A', 'B', 'C'])  
df2
```

Out[21]:

	A	B	C
3	1	2	3
AA	4	5	6
BB	7	8	9

Removendo Linhas e Colunas do DataFrame

- **Colunas:** para remover colunas, é possível remover de duas formas:
 - Removendo pelo nome da coluna;
 - Removendo pelo seu `columns` (index da coluna).
- Aqui o argumento `axis` deve ser explicitado, visto que o *default* é zero;

```
In [22]: df2.drop('A', axis=1, inplace=True)  
df2
```


Out[22]:

	B	C
3	2	3
AA	5	6
BB	8	9

```
In [23]: df2.drop(df2.columns[0], axis=1, inplace=True)  
df2
```

Out[23]:

	C
3	3
AA	6
BB	9



Cuidado! O index da coluna é obtido através de `.columns[]`, não de `.index[]`

Renomeando *index* ou *columns*

Seja o seguinte DataFrame:

```
In [24]: df3 = pd.DataFrame(data=[[1,2,3],[4,5,6],[7,8,9]], index=(0,0,1), columns=['A', 'B', 'C'])  
df3
```

Out[24]:

	A	B	C
0	1	2	3
0	4	5	6
1	7	8	9

Renomeando *index* ou *columns*

Renomeando Linhas:

```
In [25]: # Renomeie o index  
df3.rename(index={1: 'a'}, inplace=True)  
df3
```

Out[25]:

	A	B	C
0	1	2	3
0	4	5	6
a	7	8	9

Matriz original

	A	B	C
0	1	2	3
0	4	5	6
1	7	8	9

Renomeando *index* ou *columns*

Renomeando Colunas:

```
In [26]: # Defina o nome das colunas
newcols = {
    'A': 'new_column_1',
    'B': 'new_column_2',
    'C': 'new_column_3'
}
```

```
In [27]: # Use `rename()` para renomear
df3.rename(columns=newcols, inplace=True)
df3
```

Out[27]:

	new_column_1	new_column_2	new_column_3
0	1	2	3
0	4	5	6
a	7	8	9

Matriz antes de ter a coluna renomeada

	A	B	C
0	1	2	3
0	4	5	6
a	7	8	9

Formatar dados no DataFrame

Seja o seguinte DataFrame:

```
In [28]: df4 = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),  
                             index=[2.5, 12.6, 4.8], columns=[48, 49, 50])  
df4
```

Out[28]:

	48	49	50
2.5	1	2	3
12.6	4	5	6
4.8	7	8	9

Formatar dados no DataFrame

Podemos alterar elementos dentro do DataFrame das seguintes formas:

1) Alterando uma lista de elementos:

```
In [29]: # Substituir números por string  
df4.replace([1,2,3,4,5],['Awful', 'Poor', 'OK', 'Acceptable', 'Perfect'])
```

Out[29]:

	48	49	50
2.5	Awful	Poor	OK
12.6	Acceptable	Perfect	6
4.8	7	8	9

Formatar dados no DataFrame

Podemos alterar elementos dentro do DataFrame das seguintes formas:

2) Utilizando estrutura de repetição na linha:

```
In [30]: for i in range(3):  
          df4.iloc[2,i] = 0  
          df4
```

Out[30]:

	48	49	50
2.5	1	2	3
12.6	4	5	6
4.8	0	0	0

Formatar dados no DataFrame

Podemos alterar elementos dentro do DataFrame das seguintes formas:

3) Utilizando estrutura de repetição na coluna:

```
In [31]: for i in range(3):  
         df4[48] = 1  
         df4
```

Out[31]:

	48	49	50
2.5	1	2	3
12.6	1	5	6
4.8	1	0	0

Aplicando função *lambda* no DataFrame

- O que é uma função *lambda*?
 - Todas as características de uma função *lambda* são muito parecidas com as funções comuns que já vimos, com exceção de duas coisas: elas não possuem uma definição em código, ou seja, são declaradas como variáveis e não possuem um *def* próprio; e elas são funções de **uma** linha, que funcionam como se houvesse a instrução *return* antes do comando.
 - Exemplo de uma função tradicional:

```
In [32]: def timesTwo(numero):  
         return numero*2
```


Aplicando função *lambda* no DataFrame

- Seja o seguinte DataFrame:

```
In [33]: df5 = pd.DataFrame(data=[[1,2,3],[4,5,6],[7,8,9]], index=(0,1,2), columns=['A', 'B', 'C'])
```

```
In [34]: df5
```

Out[34]:

	A	B	C
0	1	2	3
1	4	5	6
2	7	8	9

Aplicando função *lambda* no DataFrame

- Para aplicarmos a função definida anteriormente em cada elemento do DataFrame, precisamos fazer da seguinte forma:

```
In [35]: for i in range(len(df5)):
          for j in range(len(df5)):
            df5.iloc[i,j] = timesTwo(df5.iloc[i,j])
```

```
In [36]: df5
```

Out[36]:

	A	B	C
0	2	4	6
1	8	10	12
2	14	16	18

Aplicando função *lambda* no DataFrame

- Entretanto, podemos definir em Python uma função Lambda da seguinte forma:

```
In [37]: # seja a seguinte função  
doubler = lambda x: x*2
```

- Para aplicá-la em cada elemento, o DataFrame do Pandas possui uma função chamada ApplyMap:

```
In [38]: df5 = df5.applymap(doubler)  
df5
```

Out[38]:

	A	B	C
0	4	8	12
1	16	20	24
2	28	32	36

Aplicando função *lambda* no DataFrame

- O DataFrame também possui uma função Apply() para aplicar apenas em elementos desejados em linhas ou colunas:

```
In [39]: # Vamos aplicar a função na coluna 'A'
df5['A'] = df5['A'].apply(doubler)
df5
```

Out[39]:

	A	B	C
0	8	8	12
1	32	20	24
2	56	32	36

```
In [40]: # Aplicando na linha de rótulo 0
df5.loc[0] = df5.loc[0].apply(doubler)
df5
```

Out[40]:

	A	B	C
0	16	16	24
1	32	20	24
2	56	32	36