



Fuctura

Módulo 3 - Data Science e Machine Learning

Profa. Jéssica Andrade



**BORA
REVISAR?**



- Oque é BIGDATA?
- Quais são os 5 V's do BIG DATA?
- Oque são HardSkills?
- Oque são SoftSkills?
- Oque é Amostragem?
- Oque é Inferencia?
- Oque é amostra?
- Viés de seleção?
- Viés de Resposta?
- Oque é StoryTelling?



Tratamento de dados com Pandas

- Atualmente a maior demanda de vagas para cientistas de dados é em tratamento de dados.
- Vamos começar a alfabetização fundamental em Pandas.
- **Entrega da semana:** o passo a passo desse material deverá ser replicado em algum conjunto de dados csv à sua escolha, e enviado por link Colab no Classroom.



O que vamos aprender hoje?

Nessa aula veremos:

- Conceitos fundamentais;
- Primeiros passos com Pandas;
- Iniciando com tratamento de dados;
- Análise exploratória de dados.



Objetivo dessa aula

- Essa aula pressupõe que você está **saindo do zero** em Pandas.
- Partiremos do início, chegando aos **principais** comandos.
- Repita o processo em outros conjuntos de dados, até ganhar confiança.





O que é o Pandas?

- Na verdade o nome vem de **Panel Data** (“Dados de Painel”)
- É uma biblioteca criada para manipular dados estruturados de forma rápida e expressiva.
- Possui ainda funcionalidades de manipulação de dados de séries temporais de alta performance (ex. dados financeiros).
- É um dos principais componentes do **portfólio Python** para **análise de dados**.
- Pandas é utilizado na fase de Preparação dos Dados (uma das fases mais importantes no processo de Data Science).



O que é o Pandas?

- Pandas contém **estruturas de alto nível** e ferramentas de **manipulação de dados** que tornam a análise desses dados mais rápida e fácil com Python.
- Funciona muito bem com NumPy.



Pandas e Numpy

Juntos eles oferecem:

- Estruturas de dados de alto nível;
- Funcionamento de séries temporais;
- A mesma estrutura de dados irá conseguir tratar dados de séries temporais e de não séries temporais.
- Tratamento flexível de dados *missing*.

(Falta de dados).



Pandas

Para se trabalhar com Pandas, devemos conhecer muito bem essas duas estruturadas de dados:

- Series
- Data Frames

| Series | | Series | | DataFrame | |
|--------|---------|--------|---------|-----------|---------|
| verde | laranja | verde | laranja | verde | laranja |
| 0 | 3 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 3 | 1 | 3 |
| 2 | 0 | 2 | 7 | 2 | 7 |
| 3 | 1 | 3 | 2 | 3 | 2 |



Pandas

Series:

é um objeto unidimensional, tipo um vetor, que contém uma sequência de objetos do mesmo tipo. Sua formação é:

```
serie = pd.Series(data, index=index)
```

Onde data(dados) pode ser uma lista, um dicionário, um array, etc, e index(índice) é uma lista de índices (se omitida, é preenchida com inteiros a partir do zero).

```
serie1 = pd.Series([10, 9, 8, 2, 5])  
serie1
```



Pandas

Series

index:

- podem ser inseridos manualmente;
- não precisam ser números inteiros;

```
serie2 = pd.Series([4, 7, 6, 3], index=['a', 'b', 'c', 'd'])  
serie2
```

- o valor que corresponde ao índice[i] pode ser acessado com series[i]

```
serie2['c']
```



Pandas

DataFrames:

Visualmente são as famosas tabelas do Pandas.

Representam uma **estrutura tabular** semelhante à estrutura de uma **planilha do Microsoft Excel**, contendo uma coleção de colunas em que **cada uma** pode ser de diferentes tipos de valores (int, float, str, etc.).

Possuem **index** e **linhas**. Os dados de um dataframe são armazenados em um ou mais blocos bidimensionais, ao invés de listas, dicionários ou alguma outra estrutura de array.



Dicas:

- **Tenha um roteiro:** muitas vezes teremos várias formas de chegar ao mesmo resultado. Para evitar “se perder” no mar de possibilidades, desenvolva o “seu jeito”, o seu passo a passo. Somente com a repetição do “seu” processo em várias análises você irá adquirir confiança para fazer esse trabalho de forma mais livre.
- **Não pule etapas:** um erro comum é querer partir para gráficos, ou mesmo para gráficos mirabolantes. O trabalho que as empresas mais precisam atualmente é o de tratamento dos dados.



Pandas

Dicas:

- **Cuidado ao trabalhar com dados sensíveis!**

Chamamos dados sensíveis os dados cujo conteúdo está amparado pela Lei Geral de Proteção de Dados Pessoais (LGPD).

[LGPD](#)

- Trabalharemos com dados públicos abertos, alguns deles sensíveis, disponibilizados publicamente por força de lei. Nas empresas, em geral os dados são obtidos por consentimento do usuário nos Termos e Condições, portanto não temos acesso a eles.



Pandas

- Vamos trabalhar em cima de um roteiro simples, claro e objetivo, repetindo alguns comandos com novos dados. É preciso aprender esse caminho, essa sequência de passos, repeti-la em outras bases de dados, para adquirir entendimento e autonomia no processo de tratamento de dados.



Parte II: Primeiros passos com Pandas





Pandas:

- Abrir novo notebook em Google colab;



```
import pandas as pd  
import numpy as np
```

- O que já sabemos?



```
#0 que já sabemos?  
serie1 =np.array([10, 9, 8, 2, 5])  
serie1
```



Pandas

- criando a primeira serie:

```
▶ serie1 = pd.Series([10, 9, 8, 2, 5])  
serie1
```

```
0    10  
1     9  
2     8  
3     2  
4     5  
dtype: int64
```



Pandas

- visualizando os valores da *serie* criada:

```
▶ serie1.values
```

```
👤 array([10,  9,  8,  2,  5])
```

- solicitando informações sobre o *index*:

```
▶ serie1.index
```

```
RangeIndex(start=0, stop=5, step=1)
```



Pandas

- Criando *series* com random:



```
serie2 = pd.Series(np.random.rand(10), index=np.arange(1, 11))  
serie2
```

Random vem de “randômico”, “randomizar”. Veremos bastante esses termos em Data Science, para referenciar valores aleatórios.



Pandas

- Criando *series* já com definição de *index*:

 `serie2 = pd.Series([4, 7, 6, 3], index=['a', 'b', 'c', 'd'])`
`serie2`

Apesar de poder ser gerado automaticamente, podemos atribuir valores ao *index*.

- Lembrando que, nas *series*, diferentemente do array Numpy, os valores podem possuir diferentes tipos, assim como o *index*.



```
serie2 = pd.Series([True, 7, 6, 3], index=[4.5, 'b', 'c', 'd'])  
serie2
```

- Seleções com auxílio do *index*:



```
serie2['c']
```



```
serie2[4.5]
```



Pandas

- Criando uma série a partir de um dicionário: vamos criar primeiramente nosso dicionário e armazenar em **capitais** e em seguida verificar o tipo:



```
capitais = {  
    'Pernambuco': 'Recife',  
    'Paraíba': 'João Pessoa',  
    'Ceará': 'Fortaleza',  
    'Bahia': 'Salvador',  
    'Alagoas': 'Maceió'  
}
```



```
type(capitais)
```



Pandas

- Agora passamos nosso dicionário como parâmetro para criarmos nossa série:

▶ serie3 = pd.Series(capitais)
serie3

▶ type(serie3)



Pandas

- Agora vamos criar uma série um pouco mais interessante.

As dez empresas mais valiosas do mundo:

1. Apple: US\$ 947,062 bilhões;
2. Google: US\$ 819,573 bilhões;
3. Amazon: US\$ 705,646 bilhões;
4. Microsoft: US\$ 611,460 bilhões;
5. Tencent: US\$ 214,023 bilhões;
6. McDonald's: US\$ 196,526 bilhões;
7. Visa: US\$ 191,032 bilhões;
8. Facebook: US\$ 186,421 bilhões;
9. Alibaba: US\$ 169,966 bilhões;
10. Louis Vuitton: US\$ 124,273 bilhões.



Pandas

- Vamos pegar essas informações e criar um dicionário:

```
▶ dictEmpresas = {'Apple': 947.062,  
                  'Google': 819.573,  
                  'Amazon': 705.646,  
                  'Microsoft': 611.460,  
                  'Tencent': 214.023,  
                  'McDonalds': 196.526,  
                  'Visa': 191.032,  
                  'Facebook': 186.421,  
                  'Alibaba': 169.966,  
                  'Louis Vuitton': 124.273  
}
```



Pandas

- Agora atribuímos esse dicionário a uma nova série:



```
serie4 = pd.Series(dictEmpresas)  
serie4
```



Pandas

- Agora vamos usar as mesmas informações para criar a série, desta vez a partir de lista e index:

```
▶ empresas = ['Apple', 'Google', 'Amazon',
               'Microsoft', 'Tencent',
               "McDonald's", 'Visa',
               'Facebook', 'Alibaba',
               'Louis Vuitton']
```

Que é exatamente a mesma coisa de:

```
▶ empresas = ['Apple', 'Google', 'Amazon', 'Microsoft', 'Tencent', "McDonald's", 'Visa', 'Facebook', 'Alibaba', 'Louis Vuitton']
```



Pandas

- Agora vamos fazer a lista com os valores, que usaremos depois como index:

```
▶ bilhoes = [947.062, 819.573, 705.646,  
             611.460, 214.023, 196.526,  
             191.032, 186.421, 169.966,  
             124.273]
```

Que é exatamente a mesma coisa de:

```
▶ bilhoes = [947.062, 819.573, 705.646, 611.460, 214.023, 196.526, 191.032, 186.421, 169.966, 124.273]
```



Pandas

- Agora, de posse das duas listas, vamos relembrar a construção de uma série:

serie = pd.Series(data, index)

- Substituindo os dados e o index pelas nossas listas, temos:



```
serie5 = pd.Series(bilhoes, index=empresas)  
serie5
```



Pandas

- Vamos fazer um **rápido spoiler sobre gráficos**, apenas por diversão, por enquanto. Para isso, vamos importar a biblioteca Matplotlib. As importações podem ser feitas no notebook na altura em que estamos mesmo:



```
import matplotlib.pyplot as plt
```

- E agora vamos digitar apenas o seguinte:



```
serie5.plot()
```



Pandas

- O que aconteceu?

A biblioteca “improvisou” e fez o que podia, já que ainda não aprendemos os detalhes para aprimorar nosso gráfico.

- Para encerrar nosso spoiler momentâneo vamos fazer mais essa:



```
serie5.plot(kind = 'bar')
```



Pandas

Bem melhor, né?

Kind significa tipo, e bar significa que estou pedindo um gráfico do tipo barras.

Assim, já conseguimos imaginar o potencial que temos nas mãos de transformar dados em informações úteis, criar gráficos impactantes e usar esse conhecimento de forma impressionante!



Pandas

- Agora vamos precisar voltar um pouco mais para a técnica. O conhecimento sobre tratamento dos dados é uma das hard skills mais disputadas quando se trata de vagas em Python.
- Vamos conhecer os famosos Data Frames, aprender a tratar um pouco os dados deles e retomaremos aos gráficos depois desta etapa.



Parte III: Iniciando com tratamento de dados





Pandas

- A formação de um Data Frame é muito semelhante à formação de uma série, com o acréscimo de colunas, já que um “df” é uma tabela. Constitui-se assim:

```
df = pd.DataFrame(data, index, columns)
```

- Lembram das nossas matrizes em Numpy?
Uma matriz tabela, ou simplesmente matriz, pode ser convertida em um Data Frame.

Pandas



- Vamos começar relembrando como criar manualmente nossa matriz:

```
▶ data = [[1, 2, 3],  
          [4, 5, 6],  
          [7, 8, 9],  
          [10, 11, 12]]
```

- Agora vamos passar essa atribuição como parâmetro:

```
▶ dados = np.matrix(data)
```

- Vamos ganhar agilidade criando a mesma matriz da seguinte forma:



```
meusDados = np.matrix('1, 2, 3; 4, 5, 6; 7, 8, 9; 10, 11, 12')  
meusDados
```

Repare que separamos as linhas por ponto e vírgula(a cada 3 valores).



Pandas

- Já que temos uma matriz (ou seja, temos linhas e colunas), ela pode ser transformada diretamente em um Data Frame:



```
pd.DataFrame(meusDados)
```

- Criamos nosso Data Frame, certo? Agora tente o seguinte:



```
meusDados
```



Pandas

- O que aconteceu?

Nada. À medida em que as “frases” de código aumentam e se tornam mais complexas, erros como esse são **muito comuns**: tratamos algum dado, linha, coluna, mas esquecemos de **atribuir** esse processamento a uma variável!

- Talvez não tenha sido difícil perceber, pela simplicidade da linha. Mas é preciso atenção.



- Dependendo da situação, pode ficar difícil ou confuso atribuir o tratamento dos dados a variáveis. Nesse caso, até ganhar segurança, você faz uma primeira linha com a operação, e na segunda, repete o processo, dessa vez com a atribuição.



Pandas

- Vamos repetir a operação,



```
pd.DataFrame(meusDados)
```

- Dessa vez armazenando o resultado:



```
meuDf = pd.DataFrame(meusDados)
```

- Agora, se verificarmos o tipo, temos:



```
type(meuDf)
```



Pandas

- Para verificarmos os tipos dos dados do nosso Data Frame:



meuDf.dtypes

- Para vermos mais informações de todo o Data Frame de forma simples, temos o .info()



meuDf.info()



Pandas

- Modificando o tipo das variáveis com **astype()**

É **muito** comum que, ao carregarmos nossas bases de dados, precisemos modificar o tipo dos dados pra conseguir trabalhar com eles.

- Modificando o tipo de dados de todo o df:



```
meuDf.astype(float)
```



Pandas

- Modificando o tipo das variáveis com `astype()`

Porém, ao digitarmos novamente



meuDf

Percebemos que os valores estão inteiros.

O que esquecemos?



Pandas

O que esquecemos?

Não armazenamos a nova informação, seja na mesma variável ou em outra. Vamos manter o mesmo nome, por enquanto:



```
meuDf = meuDf.astype(float)  
meuDf
```



Pandas

- Modificando o tipo das variáveis com **astype()**

Vamos supor que agora precisamos trabalhar com esses dados como strings apenas na coluna 1. Vamos modificar apenas o tipo dos valores dessa coluna. E dessa vez já vamos lembrar de salvar tudo isso no mesmo local, ou seja, a primeira coluna no nosso Data Frame:



```
meuDf[1] = meuDf[1].astype(str)  
meuDf
```



Pandas

- Modificando o tipo das variáveis com **astype()**

Agora, quando pedimos novamente informações sobre o tipo temos o seguinte:

```
meuDf.dtypes
0    float64
1      object
2    float64
dtype: object
```

- Um dos primeiros comandos que usamos imediatamente após carregarmos nosso Data Frame costuma ser o shape.
- Isso porque quando trabalhamos com grandes bases de dados, não é viável visualizar toda a tabela. O shape nos dá a dimensão do que estamos trabalhando:



```
meuDf.shape
```

- Também é muito comum usarmos em seguida o head, para sabermos como está visualmente nosso Data Frame. o head, por padrão, nos dá as 5 primeiras linhas, mas podemos passar um valor como parâmetro dentro dos parênteses para que ele exiba mais.



```
meuDf.head()
```



Pandas

- Como a nossa tabela por enquanto é pequena, vamos pedir as duas primeiras linhas:



```
meuDf.head(2)
```



- De forma semelhante ao head, o **tail** serve para verificarmos as últimas linhas do Data Frame. Também tem por padrão as cinco últimas, mas podemos passar outro valor por parâmetro, nos parênteses.



```
meuDf.tail()
```



```
meuDf.tail(2)
```



Pandas

- usando `.describe()`

Como mágica, digitando essas poucas letras temos um “resumo” estatístico do nosso Data Frame.



`meuDf.describe()`



- Mesmo com nosso Data Frame já criado, podemos deixá-lo mais organizado, criando seu index:

```
▶ indice = ['Linha ' + str(i) for i in range(4)]
```

- E também suas colunas:

```
▶ columnas = ['Coluna ' + str(i) for i in range(3)]
```



Pandas

- Agora que já sabemos a formação de um Data Frame (data, index, columns), montamos o nosso, de forma mais organizada:

▶ df2 = pd.DataFrame(data= meusDados, index= indice, columns= colunas)
df2

- Felizmente nosso pandinha é inteligente e podemos abreviar tudo isso em:

▶ df2 = pd.DataFrame(meusDados, indice, colunas)
df2

- Apesar de não muito intuitivo, é possível criar um Data Frame manualmente:

```
▶ df3 = {'Coluna0': {'Linha0': 1, 'Linha1': 4, 'Linha2': 7},  
         'Coluna1': {'Linha0': 2, 'Linha1': 5, 'Linha2': 8},  
         'Coluna2': {'Linha0': 3, 'Linha1': 6, 'Linha2': 9}  
     }  
  
df3
```

- E em seguida atribuímos juntamente com o índice e colunas anteriormente criados:



Pandas

- E em seguida passamos como parâmetro:



```
df3 = pd.DataFrame(df3)  
df3
```

Perceba que nesse caso modificamos o “df3”, de forma que daqui para frente ele é o novo Data Frame.



Pandas

- Vamos avançar. Agora vamos **concatenar** alguns dos Data Frames que criamos até aqui:



```
df4 = pd.concat([meuDf, df2, df3])  
df4
```

- O que aconteceu?



- A concatenação foi feita com sucesso, mas por questões de dimensões diferentes nas tabelas, algumas tuplas foram preenchidas com NaN.
- Na fase de tratamento de dados, muitas vezes encontraremos NaN (Not A Number). Fazer o tratamento desses valores é quase que corriqueiro na rotina de um cientista de dados.



Pandas

- Nessa etapa, voltamos aos conhecimentos básicos sobre **estatística**: algumas vezes será fundamental substituir valores NaN pela média. Outras vezes, isso pode ser perigoso. A depender da situação, da suas escolhas como cientista de dados, você poderá tratá-los substituindo pela média, mediana, etc, ou simplesmente excluí-los.



- Como me guiar para o melhor procedimento nesses casos?
- Primeiro você precisa saber com o quê está lidando. Ou seja, saber onde estão e quantos são os valores nulos.
- `isnull`:

Serve para mostrar os valores nulos. A resposta, nesse caso, será em bo



```
df4.isnull()
```



Pandas

- `isnull`:

Como na “vida real” teremos alguns milhões de linhas, o `isnull` sozinho nem sempre vai lhe ajudar. Mas o somatório dele, muitas vezes, é a “info” de milhões.

Para contabilizar o total de valores nulos no nosso Data Frame:



```
df4.isnull().sum()
```



Pandas

- `isnull`:

De posse da informação sobre os valores nulos, você precisará decidir. Lembre-se dos conceitos de média, moda e mediana. Deletar esses valores algumas vezes é a melhor opção. Outras vezes, isso pode nos deixar com uma quantidade muito menor de dados, o que em alguns casos pode inviabilizar o prosseguimento do seu trabalho com aquele conjunto de dados!



Pandas

Tratando dados nulos

- **fillna:** leia como um “filtrar Not A Number”

Usaremos para filtrar os valores Not a Number.



`df4.fillna`

Apesar de essa ser sua sintaxe básica, geralmente usamos parênteses em seguida, para passar outras instruções. A seguir, vamos substituir todos os valores NaN com a média dos valores válidos das colunas do Data Frame.



Pandas

- **fillna:**

Substituindo todos os valores NaN do nosso Data Frame pela média dos valores válidos das colunas:



```
df5 = df4.fillna(df4.mean())
df5
```

- **dropna:**

É o “último recurso”, quando não foi possível tratar os valores NaN ou o tratamento mais prejudicava que ajudava.



Pandas

- copiando um Data Frame:

Antes de prosseguir com qualquer comando drop, pode ser muito útil copiar o conteúdo de um Data Frame, quando queremos mexer nele com a prerrogativa de voltar a usar o original, caso precise, sem ter que carregar novamente.



```
df8 = df5.copy()
```



Pandas

- importante!

copy é usado para criar uma cópia de um objeto pandas (Dataframe, serie). As variáveis também são usadas para gerar a cópia de um objeto, mas as variáveis são apenas um ponteiro para um objeto e qualquer alteração em novos dados também alterará os dados anteriores.



Pandas

- **dropna**: é o nosso velho `.drop`, dessa vez direcionado a valores **Not A number**

Geralmente é o “último recurso”, quando não foi possível tratar os valores NaN ou o tratamento mais prejudicaria que ajudaria.

Podemos perder muitos dados com essa opção, por isso deve ser usada com cautela.



Pandas

- **dropna:**

```
▶ df4.dropna(inplace = True)  
df4
```

O que dissemos com esse comando?

“Pandas, no Data Frame df4, drop(delete) not a numbers
no lugar onde eles forem verdadeiros.



Pandas

- **dropna:**

Agora que perdemos nosso df4, vamos relembrar quem é o nosso Data Frame 5:



O que aconteceu na coluna 1?

Nesse caso, vamos optar por deletar esses valores (que levarão com ele suas informações correlatas):



```
df5.dropna(subset = [1], inplace = True)  
df5
```



Pandas

- **drop:**

Rpare que não estamos mais selecionando para deletar a partir dos valores NaN, e sim a partir de uma livre escolha nossa (por isso já não é **.dropna**). Em geral, nossas bases de dados terão inúmeras colunas, e muitas vezes algumas não nos interessam para um determinado fim. Podemos excluir colunas(ou linhas) para reduzirmos nosso Data Frame e deixá-lo mais “limpo”.



Pandas

- **drop**

Deletando colunas inteiras:

Escolhemos a “Coluna0”



```
df6 = df5.drop(columns = ['Coluna0'])  
df6
```

Deletando linhas inteiras:

Escolhemos a linha 2



```
df7 = df6.drop(index = [2])  
df7
```



Pandas

- valores únicos:

Para sabermos quais os valores únicos em uma coluna:



```
df5.Coluna2.unique()
```

Podemos usá-lo para saber as colunas ou o índice do nosso DF:



```
df5.columns.unique()
```



```
df5.index.unique()
```



Pandas

- valores únicos:

Para uma saída mais “limpa”, podemos envolver tudo isso em um print().

▶ `print(df5.Coluna2.unique())`

▶ `print(df5.columns.unique())`

▶ `print(df5.index.unique())`

Repare que, como queremos somente a informação, não precisamos armazenar o resultado.



Pandas

- usando **sort_values()**

Serve para “ordenar por”. Escolhemos o que queremos ordenar primeiramente, então, passamos nos parênteses. Desenterrando nosso df8 (que copiamos a tempo, lá atrás), temos:



```
df8.sort_values(by = 'Coluna1')
```

Aqui, escolhemos ordenar pela “Coluna1”. Muito útil, por exemplo, pra ordenar uma tabela a partir das idades.



Pandas

Renomeando colunas e linhas(index) com **df.rename()**

- **para coluna:**

Dentro dos parênteses vamos especificar primeiramente que iremos renomear algo entre as colunas, depois, entre chaves, colocamos o valor atual e o valor desejado, seguido de inplace = True

 df8.rename(columns = {'Coluna1':'COLUNA 1'}, inplace = True)
df8



Pandas

Renomeando colunas e linhas(index) com `df.rename()`

```
▶ df8.rename(columns = {'Coluna1':'COLUNA 1'}, inplace = True)  
df8
```

Lembre-se: sempre armazenar em novo Data Frame
caso deseje fixar essa mudança daqui para frente!



Pandas

Renomeando colunas e linhas(index) com **df.rename()**

- **para várias colunas:**

Na maior parte das vezes vamos fazer isso manualmente (sem o for que usamos nas Series) porque queremos nomes específicos e com boa formatação, ou seja, digitamos o novo nome de cada coluna.



Pandas

Renomeando colunas e linhas(index) com **df.rename()**

- para várias colunas:

Vamos fazer de forma semelhante ao que acabamos de fazer, apenas adicionando as outras colunas também dentro das chaves.



```
df8.rename(columns = {0:'janeiro', 1:'fevereiro', 2:'março', 'Coluna 0':'abril',
                     'Coluna 1':'maio', 'Coluna 2': 'junho', 'Coluna0':'julho',
                     'COLUNA 1': 'agosto', 'Coluna2': 'setembro'}, inplace = True)
df8
```



Renomeando colunas e linhas(index) com `df.rename()`

- **para várias linhas:**

Não tem mistério! A essa altura, o tamanho da linha de código não irá mais nos assustar. Já sabemos que ficará grande apenas porque queremos renomear todas as linhas.

Como ainda temos valores `NaN`, vamos usar `.dropna()` para excluí-los, o que também diminuirá a quantidade de linhas para renomearmos.



Pandas

- Deletando primeiramente o restante dos valores nulos



```
df8.dropna(subset = ['fevereiro'], inplace = True)  
df8
```

- Agora vamos renomear as linhas restantes:



```
df9 = df8.rename(index = {0: 'Elon Musk', 1:'Bill Gates',  
2:'Bernard Arnault', 3:'Mark Zuckerberg'})  
df9
```



Pandas

Reiniciando o index com `reset_index()`

- **para várias linhas:**

Se não for interessante ou necessário renomear linha por linha, podemos dar um “reset” no index:



```
df8.reset_index( )
```

Nesse caso você pode manter o antigo index como uma coluna ou deletá-lo, da forma como já vimos.



Pandas

- Contando valores com `.value_counts()`.



```
df8.value_counts('março')
```

- Value counts para obter o total de colunas:



```
df8.columns.value_counts().sum()
```

- Value counts para o total de linhas:



```
df8.index.value_counts().sum()
```



Próxima aula: avançando para dados abertos

