

Aula 04 (Pandas)

DATA SCIENCE IPT

TURMA 02

Pandas é uma biblioteca Python que **fornece estruturas de dados de alto desempenho** (fáceis de usar) e ferramentas de análise de dados para a linguagem de programação Python. Python com Pandas é usado em uma ampla variedade de campos, incluindo domínios acadêmicos e comerciais, incluindo finanças, economia, estatísticas, análises, etc.

Principais Estruturas de Dados em Pandas:

Series: é um array rotulado unidimensional capaz de armazenar dados de qualquer tipo (inteiro, string, float, objetos python, etc.). Os rótulos formam o índice.

DataFrame é uma estrutura de dados rotulada bidimensional com colunas de tipos potencialmente diferentes. É algo semelhante a uma planilha ou uma tabela de um banco de dados relacional. Geralmente é o objeto de pandas mais comumente usado.

Pandas Series:

Conteúdo

Criando uma série a partir de uma lista. Observando o índice original e, depois, definindo um índice de strings...

```
jupyter testdrive Last Checkpoint: 2 horas atrás (autosaved)
File Edit View Insert Cell Kernel Widgets Help
[Icons] Run Code

In [14]: import pandas as pd

In [18]: ms=pd.Series([8.4,7.3,9.2])
ms
Out[18]: 0    8.4
         1    7.3
         2    9.2
         dtype: float64

In [19]: ms.index=['João','Maria','Rafael']

In [20]: ms
Out[20]: João      8.4
         Maria     7.3
         Rafael    9.2
         dtype: float64
```

Use Tab depois do ponto '.' para ver métodos e propriedades dos objetos...
Exemplo: ms.Tab...

Atributos e métodos de um objeto (como Series, por exemplo).
Atributos são características (tamanho por exemplo), não requerem () na chamada. Métodos executam ações. Sintaticamente, exigem um () que pode conter parâmetros....
Note o uso do método max() e do atributo shape

```
In [14]: import pandas as pd
```

```
In [21]: ms=pd.Series([8.4,7.3,9.2])  
ms
```

```
Out[21]: 0    8.4  
         1    7.3  
         2    9.2  
         dtype: float64
```

```
In [23]: ms.max()
```

```
Out[23]: 9.2
```

```
In [24]: ms.shape
```

```
Out[24]: (3,)
```

Acessando elementos da Série pelo valor ou posição do índice

```
In [14]: import pandas as pd
```

```
In [34]: ms=pd.Series([8.4,7.3,9.2],index=['a','b','c'])  
ms
```

```
Out[34]: a    8.4  
        b    7.3  
        c    9.2  
        dtype: float64
```

Valor

```
In [35]: ms['b']
```

```
Out[35]: 7.3
```

Posição

```
In [36]: ms[1]
```

```
Out[36]: 7.3
```

```
In [37]: ms[['a','c']]
```

```
Out[37]: a    8.4  
        c    9.2  
        dtype: float64
```

Transformando uma coluna de um .csv em série..observando os primeiros e últimos elementos

```
In [14]: import pandas as pd
```

```
In [54]: kroton=pd.read_csv('c:\\dados\\krot3.csv',sep=';',usecols=['Fechamento'],squeeze=True)
```


```
In [56]: kroton.head()
```

```
Out[56]: 0    14.28  
1    14.08  
2    13.85  
3    13.57  
4    13.56  
Name: Fechamento, dtype: float64
```

```
In [57]: kroton.tail()
```

```
Out[57]: 172    13.60  
173    13.55  
174    14.04  
175     0.00  
176    14.04  
Name: Fechamento, dtype: float64
```

Impõe uma
dimensão..será
False em
Dataframes



Pretendíamos já colocar a coluna Data como índice, mas a coluna foi considerada inválida para índice. Nossa alternativa foi criar outra série só com a coluna Data, transformá-la em data e depois usar essa série como índice em kroton....

```
In [73]: import pandas as pd  
import matplotlib.pyplot as plt
```

```
In [250]: kroton=pd.read_csv('c:\\dados\\kroton3.csv',sep=';',usecols=['Fechamento'],squeeze=True)  
ind=pd.read_csv('c:\\dados\\kroton3.csv',sep=';',usecols=['Data'],squeeze=True)  
kroton.index=pd.to_datetime(ind)  
kroton.head()
```

```
Out[250]: Data  
2016-07-01    14.28  
2016-07-04    14.08  
2016-07-05    13.85  
2016-07-06    13.57  
2016-07-07    13.56  
Name: Fechamento, dtype: float64
```



Estatística descritiva básica do série Kroton....

```
In [69]: kroton.describe()

Out[69]: count      177.000000
         mean       13.083729
         std        4.241708
         min        0.000000
         25%       13.510000
         50%       14.180000
         75%       15.030000
         max       16.900000
         Name: Fechamento, dtype: float64
```

Temos valores zero.... Isso vai atrapalhar um gráfico, a média etc...

Estatística descritiva básica do série Kroton....vamos criar uma nova série krot1, sem os dias com cotação zero....para isso vamos usar “**boolean indexing**”



```
In [252]: krot1=kroton[kroton !=0.0]
```

```
In [253]: krot1.describe()
```

```
Out[253]: count      161.000000  
mean         14.383975  
std           0.985743  
min          12.120000  
25%          13.800000  
50%          14.310000  
75%          15.080000  
max          16.900000  
Name: Fechamento, dtype: float64
```

Vamos fazer um gráfico da cotação Krot3 no período da série usando a biblioteca matplotlib

```
In [261]: %matplotlib inline  
import matplotlib.pyplot as plt  
plt.plot(krot1)
```

```
Out[261]: [ <matplotlib.lines.Line2D at 0x1ccdb010c88> ]
```



Existe um método muito útil.....**apply**....com esse método aplicamos uma função qualquer (uma lambda, por exemplo... em todos os elementos da série)

```
In [277]: kroton.head()
```

```
Out[277]: Data
2016-07-01    14.28
2016-07-04    14.08
2016-07-05    13.85
2016-07-06    13.57
2016-07-07    13.56
Name: Fechamento, dtype: float64
```

```
In [278]: kroton.apply(lambda x:x+1).head()
```

```
Out[278]: Data
2016-07-01    15.28
2016-07-04    15.08
2016-07-05    14.85
2016-07-06    14.57
2016-07-07    14.56
Name: Fechamento, dtype: float64
```

Outro método muito útil.....**map**....com esse método, mapeamos os elementos da série para o valor de outra série (ou dicionário)..o mapeamento pode ser via função também.

```
In [79]: import pandas as pd  
s=pd.Series([1,4,6,8,0])  
s
```

```
Out[79]: 0    1  
         1    4  
         2    6  
         3    8  
         4    0  
         dtype: int64
```

```
In [81]: s1=s.map({1:20,8:100})  
s1
```

```
Out[81]: 0    20.0  
         1    NaN  
         2    NaN  
         3   100.0  
         4    NaN  
         dtype: float64
```

Ainda não fizemos uma ordenação. Nas Séries elas podem ser feitas pelo índice ou pelo valor....vamos ordenar a série kroton (sem zeros) pelo valor.

```
In [307]: krot1.sort_values().head(10)
```

```
Out[307]: Data
2016-12-16    12.12
2016-12-14    12.20
2016-12-15    12.20
2016-12-19    12.22
2016-12-20    12.28
2016-12-21    12.35
2016-12-12    12.40
2017-02-08    12.55
2017-02-07    12.89
2016-12-22    12.90
Name: Fechamento, dtype: float64
```

Obter série com os dias em que os valores superaram a média

```
In [268]: media=krot1.mean()  
          krot1[krot1 > media].head(10)
```

```
Out[268]: Data  
2016-07-12    14.63  
2016-07-13    15.03  
2016-07-14    15.45  
2016-07-15    15.64  
2016-07-18    15.80  
2016-07-19    15.74  
2016-07-20    14.95  
2016-07-21    14.88  
2016-07-22    15.08  
2016-07-25    14.81  
Name: Fechamento, dtype: float64
```


Obter série com os valores de krot3 aumentados em 10%

```
In [269]: krot1=krot1*1.1
```

```
In [270]: krot1.head()
```

```
Out[270]: Data
2016-07-01    15.708
2016-07-04    15.488
2016-07-05    15.235
2016-07-06    14.927
2016-07-07    14.916
Name: Fechamento, dtype: float64
```

Na série Kroton, trocar os valores nulos pela média sem os valores nulos

```
In [291]: media=(kroton[kroton!=0.0]).mean()
          print(media)
          kroton[kroton==0.0]=media
          kroton.mean()
```

```
14.383975155279506
```

```
Out[291]: 14.383975155279506
```

Na série Kroton sem zeros, obter a média da cotação a cada 5 dias...
Dica: usar rolling.

```
In [310]: krot1.rolling(5,min_periods=5).mean().head()
```

```
Out[310]: Data
2016-07-01      NaN
2016-07-04      NaN
2016-07-05      NaN
2016-07-06      NaN
2016-07-07    13.868
Name: Fechamento, dtype: float64
```

```
In [311]: krot1[0:5].mean()
```

```
Out[311]: 13.868
```

*Dataframe

Conteúdo

Como já falamos, Dataframe é uma estrutura de dados bidimensional. Podemos importar o dataframe do CSV com `**read_csv` (conforme o slide anterior) modificando dois parâmetros:
`use_cols` (deixar default: todas as colunas)
`squeeze` (deixar default: False (não é série))

```
In [314]: import pandas as pd
import numpy as np
df=pd.read_csv('C:\\dados\\employees.csv',index_col=['First Name'])
df.head()
```

Out[314]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							
Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

No **SAS** : *Dataframe é data set, **read_csv é feiro com PROC_IMPORT

Quando selecionamos **uma** coluna no Dataframe usando ['nome da coluna'], temos como resultado uma **série**....

```
In [315]: import pandas as pd
import numpy as np
df=pd.read_csv('C:\\dados\\employees.csv',index_col=['First Name'])
df['Team'].head()
```

```
Out[315]: First Name
Douglas      Marketing
Thomas      NaN
Maria      Finance
Jerry      Finance
Larry      Client Services
Name: Team, dtype: object
```


Quando selecionamos **várias** colunas no Dataframe usando [lista de colunas], temos como resultado um **dataframe**....

```
In [316]: import pandas as pd
import numpy as np
df=pd.read_csv('C:\\dados\\employees.csv',index_col=['First Name'])
df[['Salary','Team']].head()
```

Out[316]:

	Salary	Team
First Name		
Douglas	97308	Marketing
Thomas	61933	NaN
Maria	130590	Finance
Jerry	138705	Finance
Larry	101004	Client Services

Dataframe : Novas Colunas

Conteúdo

Para Criarmos uma nova coluna, basta atribuirmos um valor para uma coluna *não existente (a nova). No exemplo, “City” é a nova coluna, que terá sempre “Avaré”

```
In [318]: import pandas as pd
import numpy as np
df=pd.read_csv('C:\\dados\\employees.csv',index_col=['First Name'])
df['City']='Avaré'
df.head()
```

Out[318]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	City
First Name								
Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing	Avaré
Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN	Avaré
Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance	Avaré
Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance	Avaré
Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services	Avaré

* Se usarmos uma coluna existente, ela será atualizada...perigo!

Assim como podemos fazer operações em todos os elementos de uma série, nos dataframes, a ideia é a mesma...selecionamos uma coluna (é uma série!) e a operamos... a operação se espalha (“broadcast”) por toda a coluna...no exemplo, dobramos o salário de todos os funcionários.

```
In [334]: import pandas as pd
import numpy as np
df=pd.read_csv('C:\\dados\\employees.csv',index_col=['First Name'])
df['Salary']=2*df['Salary']
df.head()
```

Out[334]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							
Douglas	Male	8/6/1993	12:42 PM	194616	6.945	True	Marketing
Thomas	Male	3/31/1996	6:53 AM	123866	4.170	True	NaN
Maria	Female	4/23/1993	11:17 AM	261180	11.858	False	Finance
Jerry	Male	3/4/2005	1:00 PM	277410	9.340	True	Finance
Larry	Male	1/24/1998	4:47 PM	202008	1.389	True	Client Services

Com **dropna**, retiramos linhas (ou colunas) com algum ('any') ou ('all') todos elementos *missing*

Exemplo 1: retirar linhas (**axis=0..default**) onde **todos** elementos sejam *missing*

```
In [342]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\missing.csv')
df
```

Out[342]:

	nome	idade	salário
0	Marcos	NaN	3000.0
1	João	20.0	7000.0
2	NaN	NaN	NaN

```
In [344]: df.dropna(how='all',inplace=True)
df
```

Out[344]:

	nome	idade	salário
0	Marcos	NaN	3000.0
1	João	20.0	7000.0

Observe o
inplace=True...o
próprio dataframe
é alterado!

Com **dropna**, retiramos linhas (ou colunas) com algum ('any') ou ('all') todos elementos *missing*

Exemplo 2: retirar linhas (**axis=0..default**) onde **algum** elemento seja *missing*

```
In [345]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\missing.csv')
df
```

Out[345]:

	nome	idade	salário
0	Marcos	NaN	3000.0
1	João	20.0	7000.0
2	NaN	NaN	NaN

```
In [346]: df.dropna(how='any',inplace=True)
df
```

Out[346]:

	nome	idade	salário
1	João	20.0	7000.0

Com **dropna**, retiramos linhas (ou colunas) com algum ('any') ou ('all') todos elementos *missing*

Exemplo 3: retirar **colunas (axis=1)** onde **algum** elemento seja *missing...não sobrou nenhuma coluna!*

```
In [347]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\missing.csv')
df
```

Out[347]:

	nome	idade	salário
0	Marcos	NaN	3000.0
1	João	20.0	7000.0
2	NaN	NaN	NaN

```
In [348]: df.dropna(axis=1,how='any',inplace=True)
df
```

Out[348]:

0
1
2

Com o **dropna** retiramos linhas e colunas com missing values. Com o **fillna**, podemos trocar os NaN por valores determinados.

Exemplo 1 : trocar por 0.0 todos os valores missing

```
In [349]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\missing.csv')
df
```

Out[349]:

	nome	idade	salário
0	Marcos	NaN	3000.0
1	João	20.0	7000.0
2	NaN	NaN	NaN

```
In [350]: df.fillna(0.0)
```

Out[350]:

	nome	idade	salário
0	Marcos	0.0	3000.0
1	João	20.0	7000.0
2	0	0.0	0.0

Com o **fillna**, podemos trocar os NaN por valores determinados.

Exemplo 2 : trocar por 0.0 todos os valores missing da coluna salário

```
In [349]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\missing.csv')
df
```

Out[349]:

	nome	idade	salário
0	Marcos	NaN	3000.0
1	João	20.0	7000.0
2	NaN	NaN	NaN

```
In [352]: df['salário']=df['salário'].fillna(0.0)
df
```

Out[352]:

	nome	idade	salário
0	Marcos	NaN	3000.0
1	João	20.0	7000.0
2	NaN	NaN	0.0

Dataframe : category

Conteúdo

Quando temos uma grande coluna com poucos tipos de valores diferentes, uma boa ideia é usar `.astype` e converter a coluna para o tipo `category`. Haverá economia de espaço de armazenamento...

Parte 1 : usamos `.info()` para verificar o espaço de armazenamento

```
In [378]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv')
df.head(1)
```

Out[378]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing

```
In [377]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
First Name      933 non-null object
Gender          855 non-null object
Start Date      1000 non-null object
Last Login Time 1000 non-null object
Salary          1000 non-null int64
Bonus %        1000 non-null float64
Senior Management 933 non-null object
Team           957 non-null object
dtypes: float64(1), int64(1), object(6)
memory usage: 62.6+ KB
```

Dataframe : category

Conteúdo

Quando temos uma grande coluna com poucos tipos de valores diferentes, uma boa ideia é usar `.astype` e converter a coluna para o tipo `category`. Haverá muita economia de espaço de armazenamento...

Parte 2 : *usamos `.value_counts()` para contar valores únicos na coluna Gender*

```
In [381]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv')
df['Gender'].value_counts()
```

```
Out[381]: Female    431
Male      424
Name: Gender, dtype: int64
```

Parte 3 : *Transformamos a coluna 'Gender' em "category" usando `astype` e, finalmente, usamos `.info()` novamente, mostrando a redução de espaço (de 62kB para 56 kB)*

```
df['Gender']=df['Gender'].astype('category')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
First Name      933 non-null object
Gender          855 non-null category
Start Date      1000 non-null object
Last Login Time 1000 non-null object
Salary          1000 non-null int64
Bonus %         1000 non-null float64
Senior Management 933 non-null object
Team            957 non-null object
dtypes: category(1), float64(1), int64(1), object(5)
memory usage: 55.8+ KB
```

Dataframe : loc e iloc

Conteúdo

.loc e .iloc permitem o acesso a elementos por índice (loc) ou por “linha coluna” (iloc). Exemplo 1: salário de Maria por .loc

```
In [401]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
df.head()
```

Out[401]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							
Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

```
In [404]: df.loc['Maria','Salary']
```

```
Out[404]: First Name
Maria    130590
Maria     36067
Maria    106562
Maria    148857
Maria     96250
Maria     43455
Name: Salary, dtype: int64
```

Todas as correspondências do índice são obtidas...

Exemplo 2: salário de Maria por .iloc

```
In [405]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
df.head()
```

Out[405]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							
Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

```
In [406]: df.iloc[2,3]
```

Out[406]: 130590

Podemos fazer atribuições com loc e iloc ..exemplo
df.iloc[2,3]=80000

Podemos utilizar expressões lógicas para filtrar linhas do Dataframe.

Exemplo1:

Obter um novo dataframe (df1) com as linhas onde “Team” é “Legal”

```
In [414]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
df1=df[df["Team"]=="Legal"]
df1.head()
```

Out[414]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							
Dennis	Male	4/18/1987	1:35 AM	115163	10.125	False	Legal
Julie	Female	10/26/1997	3:19 PM	102508	12.637	True	Legal
Lois	NaN	4/22/1995	7:18 PM	64714	4.934	True	Legal
Scott	NaN	7/11/1991	6:58 PM	122367	5.218	False	Legal
Benjamin	Male	1/26/2005	10:06 PM	79529	7.008	True	Legal

Podemos utilizar expressões lógicas para filtrar linhas do Dataframe.

Exemplo2:

Obter um novo dataframe (df1) com as linhas onde “Team” é “Marketing e o salário é maior do que 80000.

```
In [419]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
filtro=(df["Team"]=="Marketing") & (df["Salary"]>80000)
df1=df[filtro]
df1.head(10)
```

Out[419]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							
Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
Matthew	Male	9/5/1995	2:12 AM	100612	13.645	False	Marketing
Charles	Male	9/14/2004	8:13 PM	107391	1.260	True	Marketing
Laura	NaN	7/19/2014	9:23 PM	140371	10.620	True	Marketing
Tina	Female	6/16/2016	7:47 PM	100705	16.961	True	Marketing
John	Male	12/23/1989	7:01 AM	80740	19.305	False	Marketing
Shirley	Female	2/28/1981	1:23 PM	113850	1.854	False	Marketing
Sean	Male	5/4/1996	8:59 PM	135490	19.934	False	Marketing
Norma	Female	2/28/1999	8:45 PM	114412	8.756	True	Marketing
Matthew	Male	7/31/2013	8:04 AM	142373	2.462	False	Marketing

A relação de pertinência “isin” facilita a criação de filtros.

Exemplo : Vamos selecionar funcionários de Team para um evento. Primeira dúvida: quais são os Team únicos?..usaremos **unique**...

```
In [428]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
df['Team'].unique()

Out[428]: array(['Marketing', nan, 'Finance', 'Client Services', 'Legal', 'Product',
'Engineering', 'Business Development', 'Human Resources', 'Sales',
'Distribution'], dtype=object)
```

Agora, montaremos uma lista de Teams a serem selecionados e faremos o filtro pela lista Usando **isin**.... Próximo slide

Filtro com **isin** (pertinência em lista no exemplo)

```
In [432]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
selec=['Marketing','Finance','Client Services','Legal','Product']
df[df['Team'].isin(selec)].head(10)
```

Out[432]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							
Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
Dennis	Male	4/18/1987	1:35 AM	115163	10.125	False	Legal
Ruby	Female	8/17/1987	4:20 PM	65476	10.012	True	Product
NaN	Female	7/20/2015	10:43 AM	45906	11.598	NaN	Finance
Julie	Female	10/26/1997	3:19 PM	102508	12.637	True	Legal
Kimberly	Female	1/14/1999	7:13 AM	41426	14.543	True	Finance
Lillian	Female	6/5/2016	6:09 AM	59414	1.256	False	Product

Podemos eliminar linhas/colunas usando o **drop**:

Exemplo 1: Eliminar as linhas de índice 'Maria'

```
In [448]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
df.drop(['Maria'],inplace=True)
df[df.index=='Maria']
```

Out[448]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							

Exemplo 1: Eliminar a coluna Salary'

```
In [452]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
df.drop(columns=['Salary'],inplace=True)
df.head(1)
```

Out[452]:

	Gender	Start Date	Last Login Time	Bonus %	Senior Management	Team
First Name						
Douglas	Male	8/6/1993	12:42 PM	6.945	True	Marketing

Podemos ordenar o dataframe por index ou por valor das colunas (nesse caso, ainda podemos ordenar por múltiplas colunas)

Exemplo 1: ordenando por índice:

```
In [33]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
df.sort_index(inplace=True)
df.head(7)
```

Out[33]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							
Aaron	Male	2/17/2012	10:20 AM	61602	11.849	True	Marketing
Aaron	Male	1/29/1994	6:48 PM	58755	5.097	True	Marketing
Aaron	Male	7/22/1990	2:53 PM	52119	11.343	True	Client Services
Aaron	NaN	1/22/1986	7:39 PM	63126	18.424	False	Client Services
Adam	Male	5/21/2011	1:45 AM	95327	15.120	False	Distribution
Adam	Male	12/24/1990	8:57 PM	110194	14.727	True	Product
Adam	Male	7/5/2007	11:59 AM	71276	5.027	True	Human Resources

Podemos ordenar o dataframe por index ou por valor das colunas (nesse caso, ainda podemos ordenar por múltiplas colunas)

Exemplo 2: ordenando por duas colunas (Team e Salary)

```
In [34]: import pandas as pd
import numpy as np
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
df.sort_values(['Team','Salary'],inplace=True)
df.head(7)
```

Out[34]:

	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
First Name							
Stephanie	Female	9/13/1986	1:52 AM	36844	5.574	True	Business Development
Scott	Male	8/20/2011	8:08 AM	37385	8.226	True	Business Development
Norma	Female	4/16/1996	5:40 AM	38872	9.302	True	Business Development
Robin	Female	10/1/2012	2:44 AM	41808	19.239	False	Business Development
Diana	Female	6/13/1994	4:21 PM	41831	4.548	False	Business Development
Bonnie	Female	12/17/1999	3:12 PM	42153	8.454	True	Business Development
Mary	Female	8/22/2010	8:03 AM	42214	17.538	True	Business Development

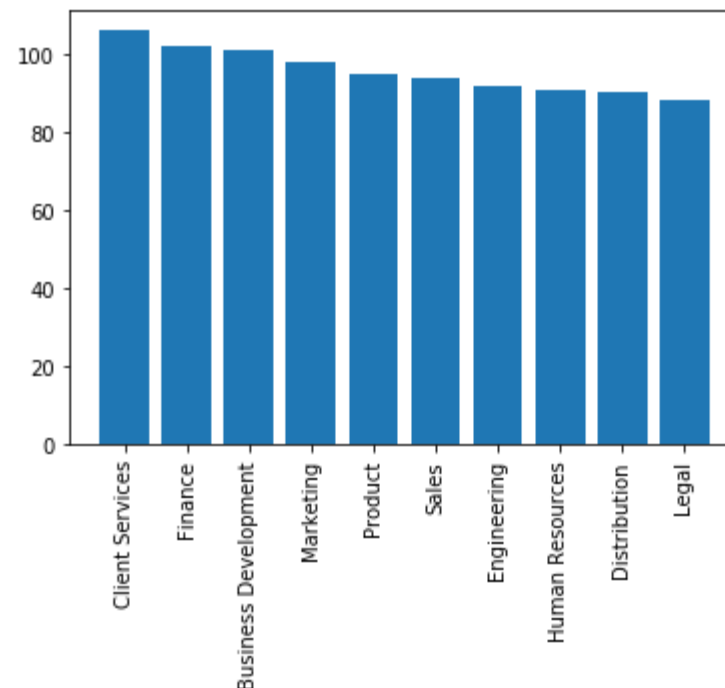
Vamos fazer um gráfico de barras com a quantidade de funcionários por time...

Inicialmente, montamos uma série (squant) com as quantidades por time. O índice dessa série traz os rótulos para xticks.

Depois, montamos o bar graph (colocando como parâmetros o índice feito com range e a própria série squant....)

```
In [470]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df=pd.read_csv('c:\\dados\\employees.csv',index_col='First Name')
squant=df['Team'].value_counts()
plt.bar(range(0,len(squant)),squant)
plt.xticks(range(0,len(squant)),squant.index, rotation=90)
plt.plot()
```

Out[470]: []



O índice hierárquico, ou multiíndice, é interessante quando queremos trabalhar com dados multidimensionais em 1D (séries) ou 2D (Dataframes), por exemplo...

```
import pandas as pd
tuples=[('Paulo',1965),('Marcelo',1965),('Marcelo',1964),('Renata',1965),('Renata',1964)]
index=pd.MultiIndex.from_tuples(tuples)
s=pd.Series([10,9,8,7,6],index=index)
s.sort_index(inplace=True)
s
```

Marcelo	1964	8
	1965	9
Paulo	1965	10
Renata	1964	6
	1965	7
dtype: int64		

Cada elemento da série é identificado por um “par” Nome-Ano. Observe a “hierarquia” do índice.

Para acessar um elemento da série, usamos, tipicamente, uma tupla com a chave....exemplo : Paulo, 1965

```
In [219]: import pandas as pd
tuples=[('Paulo',1965),('Marcelo',1965),('Marcelo',1964),('Renata',1965),('Renata',1964)]
index=pd.MultiIndex.from_tuples(tuples)
s=pd.Series([10,9,8,7,6],index=index)
s.sort_index(inplace=True)
s[('Paulo',1965)]
```

```
Out[219]: 10
```

Obviamente, o índice numérico na série continua valendo no multiindex...

```
In [220]: import pandas as pd
tuples=[('Paulo',1965),('Marcelo',1965),('Marcelo',1964),('Renata',1965),('Renata',1964)]
index=pd.MultiIndex.from_tuples(tuples)
s=pd.Series([10,9,8,7,6],index=index)
s.sort_index(inplace=True)
print(s[1])
s
```

9

```
Out[220]: Marcelo  1964      8
           1965      9
Paulo      1965     10
Renata     1964      6
           1965      7
dtype: int64
```

Também podemos fazer “slices”

```
In [221]: import pandas as pd
tuples=[('Paulo',1965),('Marcelo',1965),('Marcelo',1964),('Renata',1965),('Renata',1964)]
index=pd.MultiIndex.from_tuples(tuples)
s=pd.Series([10,9,8,7,6],index=index)
s.sort_index(inplace=True)
print(s[('Paulo',1965):('Renata',1964)])
s
```

```
Paulo    1965    10
Renata   1964     6
dtype: int64
```

```
Out[221]: Marcelo  1964     8
           1965     9
Paulo    1965    10
Renata   1964     6
           1965     7
dtype: int64
```

Tente fazer o slice sem ordenar o índice....

Analogamente, podemos fazer um multiindex para um dataframe....

```
In [230]: import pandas as pd
df=pd.read_csv('c:\\dados\\employees.csv')
#
# eliminando todos os missing, para gerar índices sem eles
#
df.dropna(inplace=True)
df.set_index(['Team', 'First Name'], inplace=True)
#
#Essencial ordenar o índice
#
df.sort_index(inplace=True)
df.head()
```

Out[230]:

		Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management
Team	First Name						
Business Development	Albert	Male	8/14/2008	5:43 PM	137840	9.705	False
	Albert	Male	9/19/1992	2:35 AM	45094	5.850	True
	Andrea	Female	1/12/2012	5:43 AM	120204	9.557	False
	Andrea	Female	10/30/1994	11:23 AM	87575	13.346	True
	Ann	Female	11/4/1984	12:17 PM	90719	6.220	False

...e fazer slice com .loc

```
In [232]: import pandas as pd
df=pd.read_csv('c:\\dados\\employees.csv')
#
# eliminando todos os missing, para gerar índices sem eles
#
df.dropna(inplace=True)
df.set_index(['Team','First Name'],inplace=True)
#
#Essencial ordenar o índice
#
df.sort_index(inplace=True)
df.loc[('Business Development','Ashley'),('Business Development','Brenda')]
```

```
Out[232]:
```

		Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	
	Team	First Name						
	Business Development	Ashley	Female	8/4/2002	11:00 AM	58698	6.811	True
		Barbara	Female	11/26/2002	5:32 AM	82884	6.837	True
		Betty	Female	6/12/2002	3:59 AM	104896	19.550	True
		Bonnie	Female	12/17/1999	3:12 PM	42153	8.454	True
		Brenda	Female	1/18/2015	4:39 PM	73749	19.332	False

...obviamente, o `.iloc` continua valendo

```
In [234]: import pandas as pd
df=pd.read_csv('c:\\dados\\employees.csv')
#
# eliminando todos os missing, para gerar índices sem eles
#
df.dropna(inplace=True)
df.set_index(['Team','First Name'],inplace=True)
#
#Essencial ordenar o índice
#
df.sort_index(inplace=True)
df.iloc[0:3,3]
```

```
Out[234]: Team          First Name
Business Development  Albert      137840
                   Albert      45094
                   Andrea      120204
Name: Salary, dtype: int64
```

Assim como temos as famosas “tabelas dinâmicas” no Excel, temos as Pivot Tables no pandas. Antes, vamos ver o método “pivot”, que transforma os valores de uma coluna em novas colunas...no nosso exemplo, essa coluna será a “Vendedor” (que só tem Celso, Ana e Daniela...próximo slide

```
In [26]: import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\loja.csv',sep=';',parse_dates=['Data'])
df.head()
```

Out[26]:

	Data	Vendedor	Valor	Produto
0	2016-01-01	Celso	9	Bola
1	2016-01-02	Celso	4	Bola
2	2016-01-03	Celso	15	Raquete
3	2016-01-04	Celso	5	Camisa
4	2016-01-05	Celso	17	Raquete

Eis o resultado do pivot, com índice data, colunas “Vendedor” e valores=Valor

```
In [29]: import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\loja.csv',sep=';',parse_dates=['Data'])
df.pivot(index='Data',columns='Vendedor',values='Valor').head()
```

Out[29]:

Vendedor	Ana	Celso	Daniela
Data			
2016-01-01	4	9	3
2016-01-02	3	4	3
2016-01-03	5	15	7
2016-01-04	3	5	3
2016-01-05	13	17	7

Pivot Table

Conteúdo

Finalmente, vamos fazer uma pivot table...primeiro no Excel

	A	B	C	D	E	F
1						
2						
3	Soma de Valor	Rótulos de Coluna				
4	Rótulos de Linha	Ana	Celso	Daniela	Total Geral	
5	Bola	66	66	64	196	
6	Camisa	62	63	40	165	
7	Raquete	87	89	94	270	
8	Total Geral	215	218	198	631	
9						
10						

Observe que, no período considerado, Daniela vendeu mais raquetes....

Agora no Pandas...observe a função de agregação (sum)

```
In [35]: import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\loja.csv',sep=';',parse_dates=['Data'])
df.pivot_table(values='Valor',index='Produto',columns='Vendedor',aggfunc=sum)
```

Out[35]:

Vendedor	Ana	Celso	Daniela
Produto			
Bola	66	66	64
Camisa	62	63	40
Raquete	87	89	94

Inicialmente, vamos entender o objeto groupby nesse exemplo:

```
In [41]: import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\employees.csv',parse_dates=['Start Date'])
grupo=df.groupby('Team')
for e in grupo:
    print(type(e),e)
```

```
<class 'tuple'> ('Business Development',      First Name  Gender Start Date Last Login Time  Salary
Bonus % \
9      Frances  Female 2002-08-08      6:51 AM  139852      7.524
33      Jean    Female 1993-12-18      9:07 AM  119082     16.180
36      Rachel  Female 2009-02-16      8:47 PM  142032     12.599
38      Stephanie Female 1986-09-13      1:52 AM   36844      5.574
41      Christine  NaN  2015-06-28      1:08 AM   66582     11.308
48      Clarence   Male 1996-03-26      5:57 AM   93581      6.083
61      Denise    Female 2001-11-06     12:03 PM  106862      3.699
```

O objeto groupby é composto de tuplas. Cada tupla traz os dados do agrupamento solicitado (Team, no caso).

Alguns métodos interessantes... `.max()` traz os valores máximos para cada coluna do grupo...

```
In [44]: import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\employees.csv',parse_dates=['Start Date'])
grupo=df.groupby('Team')
grupo.max().head()
```

Out[44]:

	Start Date	Last Login Time	Salary	Bonus %	Senior Management
Team					
Business Development	2015-06-28	9:47 AM	147417	19.626	True
Client Services	2016-02-25	9:58 PM	147183	19.894	True
Distribution	2016-05-24	9:51 AM	149105	19.908	True
Engineering	2016-03-12	9:59 AM	147362	19.850	True
Finance	2015-11-24	9:48 PM	149908	19.930	True

Alguns métodos interessantes... `.mean()` traz os valores médios para cada coluna do grupo... (numérica)

```
In [47]: import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\employees.csv',parse_dates=['Start Date'])
grupo=df.groupby('Team')
grupo.mean()
```

Out[47]:

	Salary	Bonus %
Team		
Business Development	91866.316832	10.572376
Client Services	88224.424528	10.495104
Distribution	88500.466667	9.615644
Engineering	94269.195652	10.462989
Finance	92219.480392	10.186873

Alguns métodos interessantes... `.get_group()` traz um dataframe com o grupo escolhido

```
In [46]: import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\employees.csv',parse_dates=['Start Date'])
grupo=df.groupby('Team')
grupo.get_group('Sales').head()
```

Out[46]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
13	Gary	Male	2008-01-27	11:40 PM	109831	5.831	False	Sales
35	Theresa	Female	2006-10-10	1:12 AM	85182	16.675	False	Sales
45	Roger	Male	1980-04-17	11:32 AM	88010	13.886	True	Sales
49	Chris	NaN	1980-01-24	12:13 PM	113590	3.055	False	Sales
51	NaN	NaN	2011-12-17	8:29 AM	41126	14.009	NaN	Sales

É possível fazer o groupby por mais de uma coluna....por exemplo Team and Gender..observe o método .mean() nesse caso

```
In [49]: import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\employees.csv',parse_dates=['Start Date'])
grupo=df.groupby(['Team','Gender'])
grupo.mean().head()
```

Out[49]:

		Salary	Bonus %
Team	Gender		
Business Development	Female	92669.060000	10.242480
	Male	89071.750000	10.661775
Client Services	Female	86430.083333	10.196813
	Male	93141.833333	10.831310
Distribution	Female	81328.162162	9.913135

Observe o uso de uma função no método .agg

```
In [68]: import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\employees.csv',parse_dates=['Start Date'])
grupo=df.groupby('Team')
def delta(series):
    return series.max() - series.mean()
print(grupo.agg(delta).head())
# verificando o resultado para Finance
f=grupo.get_group('Finance')
f['Salary'].max()-f['Salary'].mean()
```

	Salary	Bonus %	Senior Management
Team			
Business Development	55550.683168	9.053624	0.454545
Client Services	58958.575472	9.398896	0.580000
Distribution	60604.533333	10.292356	0.506494
Engineering	53092.804348	9.387011	0.430233
Finance	57688.519608	9.743127	0.525773

```
Out[68]: 57688.51960784313
```

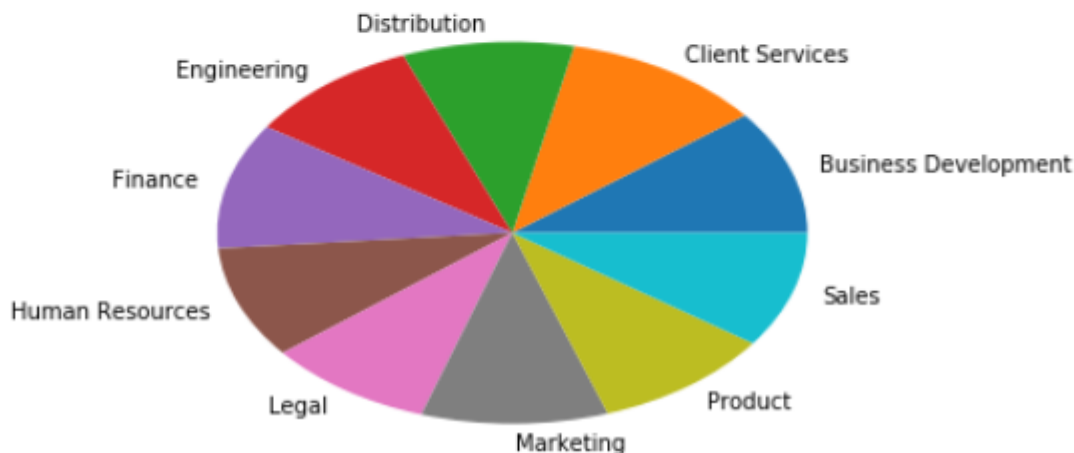
Podemos iterar pelo objeto groupby...observe o laço..cada “dados” é um dataframe

```
import pandas as pd
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\employees.csv',parse_dates=['Start Date'])
grupo=df.groupby('Team')
for setor,dados in grupo:
    print(setor)
    print(dados['Salary'].max())
```

```
Business Development
147417
Client Services
147183
Distribution
149105
Engineering
147362
Finance
149908
Human Resources
149903
Legal
148985
Marketing
```

O gráfico “pie” é muito simples. Pode receber, por exemplo, uma lista de valores e rótulos e já mostrar as partições...

```
In [83]: import pandas as pd
import matplotlib.pyplot as plt
#Observe o parse_dates que já transforma a data em objeto datetime
df=pd.read_csv('c:\\dados\\employees.csv',parse_dates=['Start Date'])
grupo=df.groupby('Team')
funcs=[] #quantidade de funcionários
setorx=[]
for setor,dados in grupo:
    setorx.append(setor)
    funcs.append(len(dados))
plt.pie(funcs,labels=setorx)
plt.show()
```



Com Pandas, podemos facilmente importar abas (sheets) do Excel como DataFrames...no exemplo, são criados 4 DataFrames a partir de um workbook (vendas.xlsx)

```
In [239]: vendas1=pd.read_excel('c:\\dados\\vendas.xlsx',sheet_name='vendas_sede')
vendas2=pd.read_excel('c:\\dados\\vendas.xlsx',sheet_name='vendas_filial')
clientes=pd.read_excel('c:\\dados\\vendas.xlsx',sheet_name='clientes')
produtos=pd.read_excel('c:\\dados\\vendas.xlsx',sheet_name='produtos')
vendas1.head(1)
```

```
Out[239]:
```

	id_cliente	id_produto	quant
0	1	503	1

```
In [240]: vendas2.head(1)
```

```
Out[240]:
```

	id_cliente	id_produto	quant
0	4	501	3

```
In [241]: clientes.head(1)
```

```
Out[241]:
```

	id	nome	cpf
0	1	João Oliveira	27685285265

```
In [242]: produtos.head(1)
```

```
Out[242]:
```

	sku	nome	descricao	preco
0	500	Bola	Esfera para esportes	100

Os DataFrames `vendas_sede` e `vendas_filial` têm as mesmas colunas. Podemos criar um novo dataframe “concatenando” os dois nas linhas (`axis=0`). Observe a transição de `vendas_sede` para `vendas_filial`...o índice de ambos foi mantido.

```
In [245]: vendas=pd.concat([vendas1,vendas2],axis=0)  
vendas.iloc[12:18]
```

Out[245]:

	id_cliente	id_produto	quant
12	2	500	1
13	5	501	3
14	6	504	2
0	4	501	3
1	15	500	3
2	15	502	1

Porém, se usarmos a opção `ignore_index=True`, um novo índice é gerado....

```
In [246]: vendas=pd.concat([vendas1,vendas2],axis=0,ignore_index=True)  
vendas.iloc[12:18]
```

Out[246]:

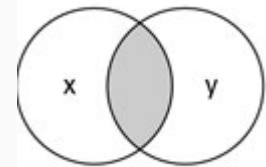
	id_cliente	id_produto	quant
12	2	500	1
13	5	501	3
14	6	504	2
15	4	501	3
16	15	500	3
17	15	502	1

Merge

Conteúdo

Vamos fazer um “**inner join**” entre os DataFrames vendas1 e vendas2 na coluna id_cliente. Assim, só ficarão as linhas com id_cliente comum aos dois DataFrames (intersecção)

how='inner'



natural join

```
In [265]: vendas1.merge(vendas2,how='inner',on='id_cliente')
```

```
Out[265]:
```

	id_cliente	id_produto_x	quant_x	id_produto_y	quant_y
0	2	505	2	501	2
1	2	505	2	505	1
2	2	505	2	504	2
3	2	500	1	501	2
4	2	500	1	505	1
5	2	500	1	504	2
6	2	500	1	501	2
7	2	500	1	505	1
8	2	500	1	504	2
9	9	504	1	504	1

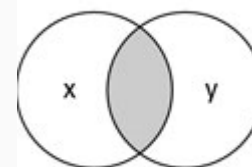
right

left

Usando conjuntos, mostre que os `id_cliente` comuns aos DataFrames `vendas1` e `vendas2` são só 2 e 9.

No slide anterior, poderíamos ter utilizado “suffixes”, para deixar mais claro o entendimento das colunas geradas

how='inner'



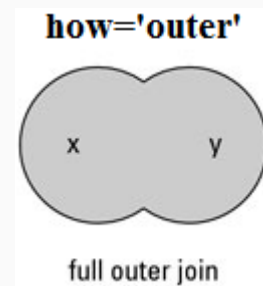
natural join

```
In [278]: vendas1.merge(vendas2,how='inner',on='id_cliente',suffixes=['-vendas1','-vendas2'])
```

Out[278]:

	id_cliente	id_produto-vendas1	quant-vendas1	id_produto-vendas2	quant-vendas2
0	2	505	2	501	2
1	2	505	2	505	1
2	2	505	2	504	2
3	2	500	1	501	2
4	2	500	1	505	1
5	2	500	1	504	2
6	2	500	1	501	2
7	2	500	1	505	1
8	2	500	1	504	2
9	9	504	1	504	1

Neste slide, faremos um **outer join** entre vendas1 e vendas2. Observe o útil parâmetro “indicator”.....



```
In [276]: outer=vendas1.merge(vendas2,how='outer',on='id_cliente',indicator=True)
          outer.head()
```

Out[276]:

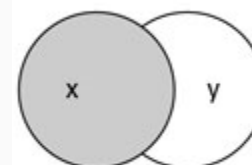
	id_cliente	id_produto_x	quant_x	id_produto_y	quant_y	_merge
0	1	503.0	1.0	NaN	NaN	left_only
1	1	500.0	3.0	NaN	NaN	left_only
2	2	505.0	2.0	501.0	2.0	both
3	2	505.0	2.0	505.0	1.0	both
4	2	505.0	2.0	504.0	2.0	both

Merge

Conteúdo

Agora, faremos um **left join** entre vendas1 e vendas2.

how='left'



left outer join

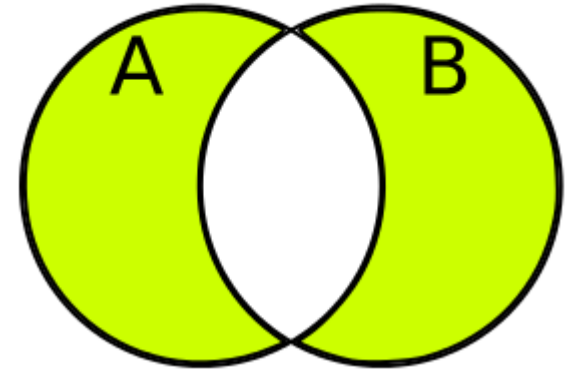
```
In [286]: left=vendas1.merge(vendas2,how='left',on='id_cliente',suffixes=['-vendas1','-vendas2'],indicator=True)
left.head(10)
```

Out[286]:

	id_cliente	id_produto-vendas1	quant-vendas1	id_produto-vendas2	quant-vendas2	_merge
0	1	503	1	NaN	NaN	left_only
1	2	505	2	501.0	2.0	both
2	2	505	2	505.0	1.0	both
3	2	505	2	504.0	2.0	both
4	3	501	2	NaN	NaN	left_only
5	2	500	1	501.0	2.0	both
6	2	500	1	505.0	1.0	both
7	2	500	1	504.0	2.0	both
8	1	500	3	NaN	NaN	left_only
9	5	500	1	NaN	NaN	left_only

Obtenha o outer join (vendas1 e vendas2) e retire a intersecção (“inner join”)

[Exclusive] Full Join ($A \oplus B$)



Incluir os dados dos compradores
no DataFrame vendas1, gerando
vendas1_comp

Merge : Exercícios

Conteúdo

```
In [306]: vendas1_comp=vendas1.merge(clientes,how='inner',left_on='id_cliente',right_on='id')  
vendas1_comp.head()
```

Out[306]:

	id_cliente	id_produto	quant	id	nome	cpf
0	1	503	1	1	João Oliveira	27685285265
1	1	500	3	1	João Oliveira	27685285265
2	2	505	2	2	Celso Freitas	77020340257
3	2	500	1	2	Celso Freitas	77020340257
4	2	500	1	2	Celso Freitas	77020340257

Incluir os dados dos produtos no DataFrame vendas (vendas1 e vendas2 concatenados), gerando vendas_prod

Merge : Exercícios

Conteúdo

```
In [311]: vendas=pd.concat([vendas1,vendas2],ignore_index=True)
vendas_prod=vendas.merge(produtos,left_on='id_produto',right_on='sku')
vendas_prod.head()
```

Out[311]:

	id_cliente	id_produto	quant	sku	nome	descricao	preco
0	1	503	1	503	Geladeira	refrigerador doméstico	2000
1	10	503	1	503	Geladeira	refrigerador doméstico	2000
2	2	505	2	505	Boné	Protetor do sol	30
3	6	505	1	505	Boné	Protetor do sol	30
4	7	505	2	505	Boné	Protetor do sol	30

Para um id de cliente :

- 1) Apresente os dados do cliente
- 2) Apresente produtos comprados em vendas (1+2) com informações de produtos e inclusão de total (quant x preço...nova coluna)
- 3) Apresente o valor total das compras do cliente

Merge : Exercícios

Conteúdo

```
In [336]: id=8
print(clientes[clientes['id']==8])
vendas=pd.concat([vendas1,vendas2],ignore_index=True)
vendas=vendas[vendas['id_cliente']==id]
vendas_prod=vendas.merge(produtos,left_on='id_produto',right_on='sku',how='inner')
total_unit=vendas_id['quant']*vendas_id['preco']
vendas_id['total']=total_unit
print(vendas_id)
print('Valor Total: ',vendas_id['total'].sum())
```

```
   id  nome      cpf
7  8  Marcelo Rezende  87279683560
   id_cliente  id_produto  quant  sku  nome \
8           8           504      2  504  Árvore de Natal
9           8           501      1  501  Prato

   descricao  preco  total
8  Decorador natalino    800    1600
9  Objeto para alimentação    20     20
Valor Total:  1620
```

O Python possui uma biblioteca para datas e horas. Essa biblioteca (Datetime) **não** é incluída “por default”. Vamos começar com o objeto “básico” **date**...

```
In [338]: import pandas as pd
import datetime as dt
meu_niver=dt.date(2018,7,6)
str(meu_niver)
```

```
Out[338]: '2018-07-06'
```

...e depois, o **datetime**....

```
In [340]: import pandas as pd
import datetime as dt
reveillon=dt.datetime(2019,1,1,0,0)
str(reveillon)
```

```
Out[340]: '2019-01-01 00:00:00'
```

Os objetos date e datetime possuem muitos métodos interessantes. Entre eles, strftime e strptime. O primeiro gera o string formatado da data conforme formato (veja formatos no próximo slide). O segundo transforma um string em objeto datetime com base em um formato.

```
In [342]: import pandas as pd
import datetime as dt
reveillon=dt.datetime(2019,1,1,0,0)
output=reveillon.strftime('%B, %d, %Y')
output
```

```
Out[342]: 'January, 01, 2019'
```

Observe, no código abaixo, a data escrita no formato usual do Brasil (dia/mês/Ano)..sendo convertido corretamente para um datetime...

```
In [343]: import pandas as pd
import datetime as dt
reveillon=dt.datetime(2019,1,1,0,0)
data=dt.datetime.strptime('30/04/2019', '%d/%m/%Y')
data
```

```
Out[343]: datetime.datetime(2019, 4, 30, 0, 0)
```

Dates and Times (formats)

Conteúdo

Directive	Description	Example	Directive	Description	Example
%a	Weekday, short version	Wed	%M	Minute 00-59	41
%A	Weekday, full version	Wednesday	%S	Second 00-59	8
%w	Weekday as a number 0-6, 0 is Sunday	3	%f	Microsecond 000000-999999	548513
%d	Day of month 01-31	31	%z	UTC offset	100
%b	Month name, short version	Dec	%Z	Timezone	CST
%B	Month name, full version	December	%j	Day number of year 001-366	365
%m	Month as a number 01-12	12	%U	Week number of year, Sunday as the first day of week, 00-53	52
%y	Year, short version, without century	18	%W	Week number of year, Monday as the first day of week, 00-53	52
%Y	Year, full version	2018	%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%H	Hour 00-23	17	%x	Local version of date	12/31/18
%I	Hour 00-12	5	%X	Local version of time	17:41:00
%p	AM/PM	PM	%%	A % character	%

É bastante comum criarmos um índice para uma série ou DataFrame composto por objetos datetime. Para isso usamos **DatetimeIndex**. Observe que usamos um método do Pandas (**pd**) e não do datetime (**dt**). Com os parâmetros disponíveis, são muitas as possibilidades. No exemplo, criamos um índice que começa em 2/12/18 e avança uma semana por vez, em 10 períodos...

```
In [345]: import pandas as pd
import datetime as dt
indice=pd.DatetimeIndex(start='2018-12-02',periods=10,freq='W')
indice

Out[345]: DatetimeIndex(['2018-12-02', '2018-12-09', '2018-12-16', '2018-12-23',
                        '2018-12-30', '2019-01-06', '2019-01-13', '2019-01-20',
                        '2019-01-27', '2019-02-03'],
                        dtype='datetime64[ns]', freq='W-SUN')
```

Quando subtraímos dois objetos datetime, obtemos um objeto **timedelta**...observe que podemos somar um timedelta a uma data...

```
In [347]: import pandas as pd
import datetime as dt
data1=dt.datetime(2018,12,3)
data2=dt.datetime(2018,12,25)
dif=data2-data1
type(dif)
```

```
Out[347]: datetime.timedelta
```

```
In [348]: dt.date(2019,1,1)+dif
```

```
Out[348]: datetime.date(2019, 1, 23)
```

O Pandas traz o “**dt accessor**”, que permite aplicar em uma **série inteira** um método de data/tempo.....observe :

- 1)Uso do pd.to_datetime (com format)
- 2)Criação da série sd
- 3)Aplicação do método day_name() a toda a série

```
In [354]: import pandas as pd
import datetime as dt
l=['2/12/2018','3/12/2018','4/12/2018','5/12/2018']
dl=pd.to_datetime(l,format='%d/%m/%Y')
sd=pd.Series(dl)
sd=sd.dt.day_name()
sd
```

```
Out[354]: 0      Sunday
1      Monday
2      Tuesday
3      Wednesday
dtype: object
```

Além do dt accessor, o Pandas traz também um accessor para strings...o str accessor...Vamos a um exemplo...o método `capitalize()` foi aplicado a toda a série (ss).

```
In [358]: import pandas as pd
import datetime as dt
l=['marcelo','maria','adriana','antonio']
ss=pd.Series(l)
ss.str.capitalize()
```

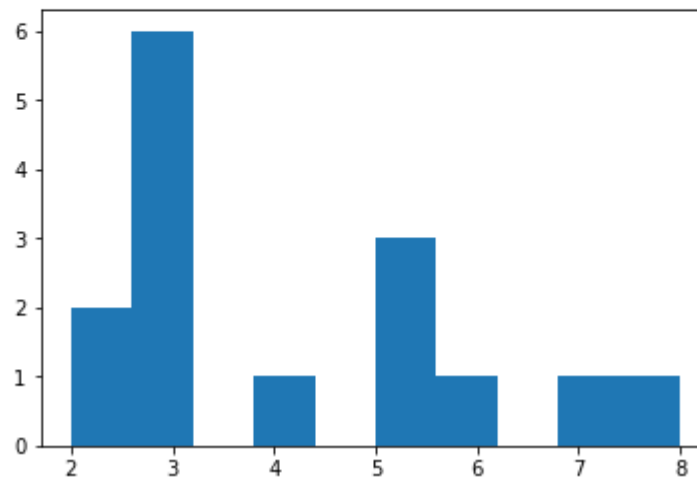
```
Out[358]: 0    Marcelo
1      Maria
2    Adriana
3    Antonio
dtype: object
```

Histogramas

Conteúdo

Com o matplotlib, é simples criar um histograma. No exemplo abaixo, passamos uma lista de inteiros...pronto!

```
In [363]: import pandas as pd  
import datetime as dt  
import matplotlib.pyplot as plt  
plt.hist([2,2,3,4,5,6,7,8,5,5,3,3,3,3])  
plt.show()
```





Cursos com Alta Performance de
Aprendizado

2019