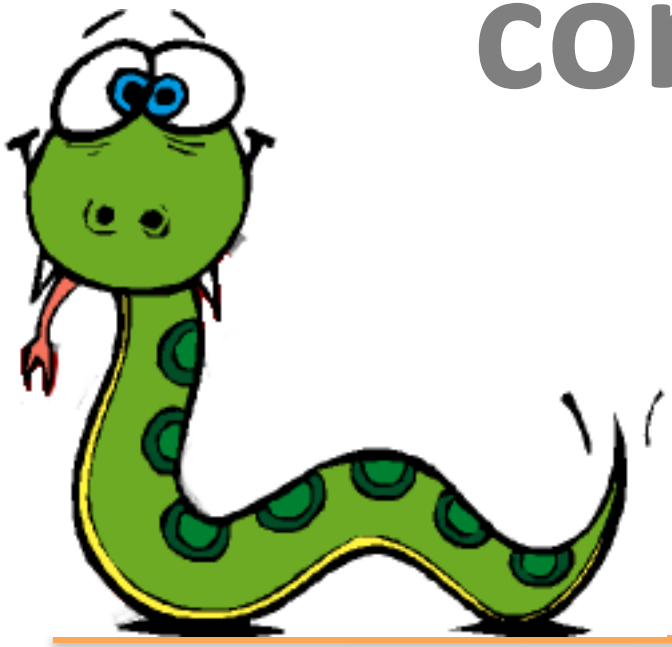
Análise de dados com Pandas e Seaborn

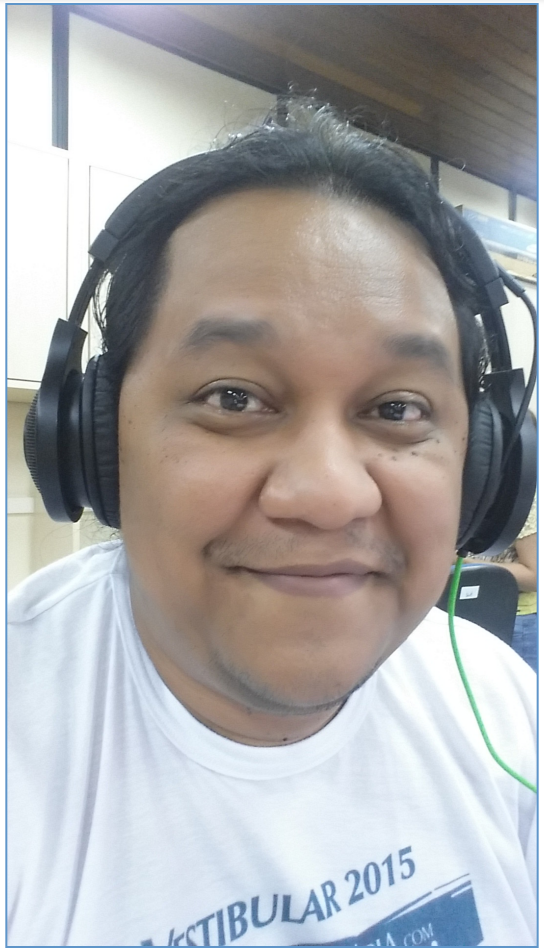


Márcio Palheta, M.Sc.
marcio.palheta@gmail.com

Apresentação

- Programador desde 2000
- Professor de programação desde 2009
- Mestre pelo ICOMP/UFAM – 2013
- Fundador da Buritech – 2014
- Doutorando pelo ICOMP/UFAM
- Pesquisador das áreas: Banco de Dados, Recuperação da Informação, Big Data, Mineração de dados e Aprendizado de Máquina

https://sites.google.com/site/marciopalheta



Agenda

- Entendendo a biblioteca
- Séries, DataFrame, pydataset e db.py
- Cargas de arquivos CSV e Excel
- Filtros e seleções
- Dados categóricos, periódicos e séries temporais
- Visualização em gráficos

Começando com o Pandas

Python Data Analysis Library

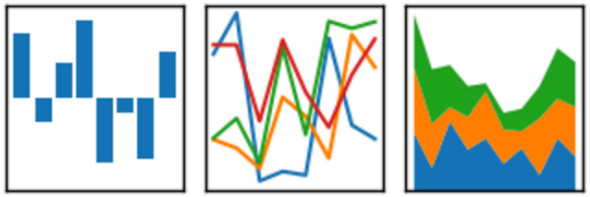
Márcio

pandas.pydata.org

Fork me on GitHub

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



overview // get pandas // documentation // community // talks // donate

Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NUMFocus sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project, and makes it possible to donate to the project.

A Fiscally Sponsored Project of

NUMFOCUS

OPEN CODE = BETTER SCIENCE

VERSIONS

Release
0.20.1 - May 2017
download // docs // pdf

Development
0.21.0 - 2017
github // docs

Previous Releases
0.19.2 - download // docs // pdf
0.18.1 - download // docs // pdf
0.17.1 - download // docs // pdf
0.16.2 - download // docs // pdf

Entendendo o contexto

- Pandas é a principal biblioteca para realização de análise de dados
- Usa o numpy para processamento de grandes massas de dados tabulados (2d)
- Oferece ferramentas para facilitar o processo de análise dos dados
- Instalação: sudo pip install pandas

Exercício 01: Séries de dados

Análise de dados

```
import pandas as pd

# Série é um array de uma dimensão
serie = pd.Series([8, 2, 7, 6, 1, 5])
serie
```

```
0    8
1    2
2    7
3    6
4    1
5    5
dtype: int64
```

Exercício 02: Parece lista

```
In [10]: # Acesso a um índice
         serie[0]

Out[10]: 8

In [13]: # Acesso com slicing
         serie[-2:]

Out[13]: 4    1
         5    5
         dtype: int64

In [14]: serie[1::2]

Out[14]: 1    2
         3    6
         5    5
         dtype: int64
```

Exercício 03: Descrição da série

```
In [21]: # Estatísticas da serie
         serie.describe()

Out[21]: count    6.000000
         mean     4.833333
         std      2.786874
         min      1.000000
         25%      2.750000
         50%      5.500000
         75%      6.750000
         max      8.000000
         dtype: float64

In [19]: # media
         serie.mean()

Out[19]: 4.833333333333333

In [20]: # mediana
         serie.median()

Out[20]: 5.5
```

Exercício 04: Dados duplicados

```
serie = pd.Series([8, 6, 7, 6, 1, 5, 6])
# Verificacao de dados duplicados
serie.duplicated()
```

```
0    False
1    False
2    False
3     True
4    False
5    False
6     True
dtype: bool
```

Exercício 05: append()

```
In [474]: # Concatenando séries com append
          serie = pd.Series([8, 6, 7, 6])
          serie2 = pd.Series([1, 3, 2])

          serie.append(serie2)

Out[474]: 0    8
          1    6
          2    7
          3    6
          0    1
          1    3
          2    2
          dtype: int64
```

DataFrames

- É a principal estrutura do curso, por suportar o processamento de matrizes
- Aceita uma lista de listas
- Internamente, é uma lista de séries
- Podemos nomear as colunas
- Não temos acesso por índice, mas sim por nome de coluna

Exercício 06: DataFrame

DataFrames

```
import pandas as pd

medias = pd.DataFrame([['Maria', 10.0], ['Ana', 8.5], ['Fernanda', 9.5]])
# numeros de linhas e colunas
medias.shape
```

```
(3, 2)
```

```
# Imprimindo o DataFrame
medias
```

	0	1
0	Maria	10.0
1	Ana	8.5
2	Fernanda	9.5

Exercício 07: nomes de colunas

```
import pandas as pd

# Trabalhando com nomes de colunas
medias = pd.DataFrame([['Maria', 10.0], ['Ana', 8.5],
                       ['Fernanda', 9.5]],
                      columns=['Nome do aluno', 'Media anual'])

medias
```

	Nome do aluno	Media anual
0	Maria	10.0
1	Ana	8.5
2	Fernanda	9.5

Exercício 08: Acesso à coluna

	Nome do aluno	Media anual
0	Maria	10.0
1	Ana	8.5
2	Fernanda	9.5

```
# Acesso a uma determinada coluna
medias['Media anual']
```

```
0     10.0
1      8.5
2      9.5
Name: Media anual, dtype: float64
```

Exercício 09: estatística

```
In [40]: medias
```

Out[40]:

	Nome do aluno	Media anual
0	Maria	10.0
1	Ana	8.5
2	Fernanda	9.5

```
In [44]: # Calculo da media da turma
         medias['Media anual'].mean()
```

```
Out[44]: 9.333333333333339
```

```
In [45]: # Mediana
         medias['Media anual'].median()
```

```
Out[45]: 9.5
```

Exercício 10: pegando a linha

Out[47]:

	Nome do aluno	Media anual
0	Maria	10.0
1	Ana	8.5
2	Fernanda	9.5

In [50]:
```
# Acesso à linha 2
medias.iloc[2]
```

Out[50]:
```
Nome do aluno    Fernanda
Media anual           9.5
Name: 2, dtype: object
```

In [51]:
```
medias.iloc[2]['Nome do aluno']
```

Out[51]:
```
'Fernanda'
```

In [52]:
```
medias.iloc[2]['Media anual']
```

Out[52]:
```
9.5
```

Pandas index

- Index é a estrutura do pandas usada para mapear um elemento para um valor único
- Lembra o trabalho realizado por chaves primárias em SGBDs
- Por padrão, é adotado o índice numérico

Exercício 11: estados

```
import pandas as pd

# Trabalhando com nomes de colunas
estados = pd.DataFrame([
['AC', 'Acre', 'Rio Branco'], ['AL', 'Alagoas', 'Maceió'],['AP', 'Amapá', 'Macapá'],
['AM', 'Amazonas', 'Manaus'],['BA', 'Bahia', 'Salvador'],['CE', 'Ceara', 'Fortaleza'],
['DF', 'Distrito Federal', 'Brasília'],['ES', 'Espírito Santo', 'Vitória'],
['GO', 'Goiás', 'Goiânia'], ['MA', 'Maranhão', 'São Luiz'], ['MT', 'Mato Grosso', 'Cuiabá'],
['MS', 'Mato Grosso do Sul', 'Campo Grande'], ['MG', 'Minas Gerais', 'Belo Horizonte'],
['PA', 'Pará', 'Belém'],['PB', 'Paraíba', 'João Pessoa'],['PR', 'Paraná', 'Curitiba'],
['PE', 'Pernambuco', 'Recife'],['PI', 'Piauí', 'Terezina'],
['RJ', 'Rio de Janeiro', 'Rio de Janeiro'],['RN', 'Rio Grande do Norte', 'Natal'],
['RS', 'Rio Grande do Sul', 'Porto Alegre'],['RO', 'Rondônia', 'Porto Velho'],
['RR', 'Roraima', 'Boa Vista'],['SC', 'Santa Catarina', 'Florianópolis'],
['SP', 'São Paulo', 'São Paulo'],['SE', 'Sergipe', 'Aracajú'],
['TO', 'Tocantins', 'Palmas']], columns=['Sigla', 'Estado', 'Capital'])
estados
```

	Sigla	Estado	Capital
0	AC	Acre	Rio Branco
1	AL	Alagoas	Maceió
2	AP	Amapá	Macapá

https://gist.github.com/marciopalheta/bbe68435d2eedd733a3b5739afb97eb4

Exercício 12: nome do estado

```
In [3]: print estados.index
        estados['Estado']
```

```
RangeIndex(start=0, stop=27, step=1)
```

```
Out[3]: 0                    Acre
        1                 Alagoas
        2                   Amapá
        3                Amazonas
        4                   Bahia
        5                   Ceara
        6        Distrito Federal
        7          Espírito Santo
        8                   Goiás
        9                Maranhão
        10           Mato Grosso
```

Exercício 13: Acesso ao índice

```
In [9]: # Acesso ao índice ZERO,
        # mas o índice poderia ser String
        estados.ix[3]

Out[9]: Sigla              AM
        Estado       Amazonas
        Capital        Manaus
        Name: 3, dtype: object

In [10]: # Só aceita valor numérico
         estados.iloc[3]

Out[10]: Sigla              AM
         Estado       Amazonas
         Capital        Manaus
         Name: 3, dtype: object
```

Exercício 14: diferença

In [12]:
```
# Limite superior fechado
estados.ix[3:6]
```

Out[12]:

	Sigla	Estado	Capital
3	AM	Amazonas	Manaus
4	BA	Bahia	Salvador
5	CE	Ceara	Fortaleza
6	DF	Distrito Federal	Brasília

In [13]:
```
# Limite superior aberto
estados.iloc[3:6]
```

Out[13]:

	Sigla	Estado	Capital
3	AM	Amazonas	Manaus
4	BA	Bahia	Salvador
5	CE	Ceara	Fortaleza

Exercício 15: troca de índices

In [23]:
```
# Alterando os índices
estados.index = pd.Index(range(10, 37))
estados
```

Out[23]:

	Sigla	Estado	Capital
10	AC	Acre	Rio Branco
11	AL	Alagoas	Maceió
12	AP	Amapá	Macapá
13	AM	Amazonas	Manaus
14	BA	Bahia	Salvador
15	CE	Ceara	Fortaleza
16	DF	Distrito Federal	Brasília

Exercício 16: nova troca

In [27]:
```
# Trocando de número para Sigla
estados.index = estados['Sigla']
estados
```

Out[27]:

	Sigla	Estado	Capital
Sigla			
AC	AC	Acre	Rio Branco
AL	AL	Alagoas	Maceió
AP	AP	Amapá	Macapá
AM	AM	Amazonas	Manaus
BA	BA	Bahia	Salvador

Exercício 17: Novo índice

```
In [29]: estados.ix['AM']

Out[29]: Sigla              AM
         Estado       Amazonas
         Capital         Manaus
         Name: AM, dtype: object
```

Exercício 18: removendo coluna

In [32]:
```
del estados['Sigla']
estados
```

Out[32]:

	Estado	Capital
Sigla		
AC	Acre	Rio Branco
AL	Alagoas	Maceió
AP	Amapá	Macapá
AM	Amazonas	Manaus
BA	Bahia	Salvador

Facilitando a carga de dados

- Até aqui, trabalhamos com dados que definimos estaticamente
- Contudo, no dia-a-dia, é comum a carga de dados a partir de arquivos XLS ou CSV
- O pandas oferece os métodos para leitura: pd.read_csv e pd.read_xls
- O read_xls requer: pip install xlrd

Exercício 19: pd.read_csv()

```
import pandas as pd
#Informações sobre terrenos de Copacabana
copacabana = pd.read_csv('copacabana.csv', delimiter=';')
copacabana.head()
```

	Posicao	Quartos	Vagas	DistIpanema	DistPraia	DistFavela	RendaMedia
0	1	3.0	0.01	1144	311	146	969501
1	0	2.0	0.01	2456	502	254	1472861
2	0	2.0	0.01	2448	772	229	1803724
3	0	2.0	0.01	1615	428	310	1124331
4	0	2.0	1.00	2358	586	287	1165764

Exercício 20: pd.read_excel()

```
import pandas as pd

# Informações sobre municípios do Amazonas
censo_amazonas = pd.read_excel('censo_ibge_amazonas.xls')
censo_amazonas.head()
```

	Código	Municípios	Gentílico	População 2010	Área territorial 2015 (km²)
0	1300029	Alvarães	alvarãense	14088	5923.46
1	1300060	Amaturá	amaturaense	9467	4754.11
2	1300086	Anamã	anamãense	10214	2453.94
3	1300102	Anori	anoriense	16317	6036.36
4	1300144	Apuí	apuiense	18007	54244.92

Exercício 21: detalhes

```
import pandas as pd

#Informações sobre terrenos de Copacabana
copacabana = pd.read_csv('copacabana.csv', delimiter=';')
copacabana.columns
```

```
Index([u'Posicao', u'Quartos', u'Vagas', u'DistIpanema', u'DistPraia',
       u'DistFavela', u'RendaMedia', u'RendaMovel', u'RendaMovelRua',
       u'Vu2009', u'Mes', u'Idade', u'Tipologia', u'AreaConstruida',
       u'VAL_UNIT', u'X', u'Y'],
      dtype='object')
```

```
copacabana['Quartos'].describe()
```

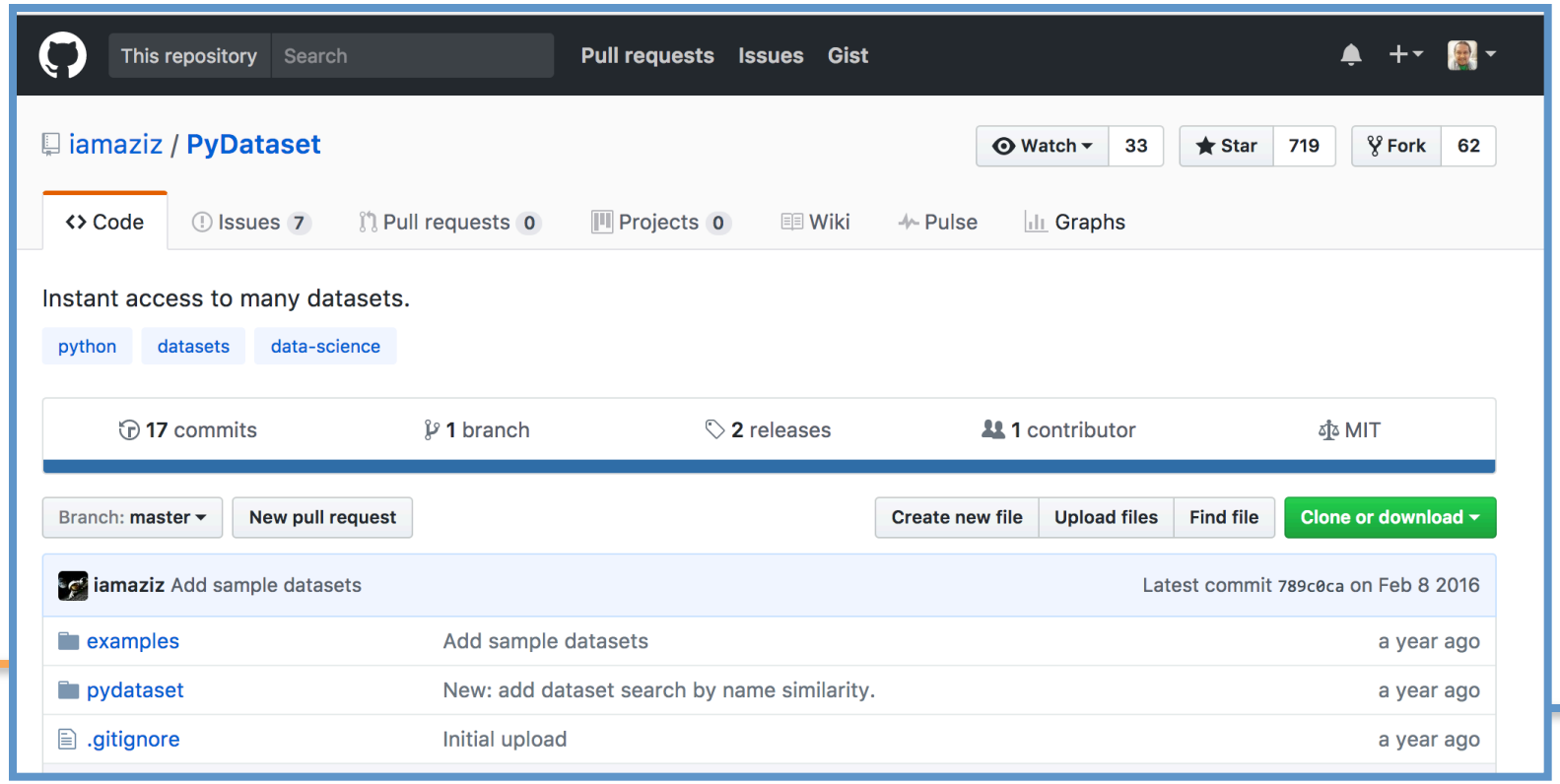
```
count    1675.000000
mean        1.767510
std         1.142523
min         0.010000
25%         1.000000
50%         2.000000
75%         3.000000
max         6.000000
Name: Quartos, dtype: float64
```

Carga de dados com pydataset

- Um dos grandes desafios da nossa área de pesquisa em Aprendizado de Máquina e Mineração de Dados é a base de dados
- Pydataset é uma biblioteca que usa o Pandas para disponibilizar acesso a 752 bases de dados públicas e conhecidas da comunidade

Carga de dados com pydataset

- $ pip install pydataset
- https://github.com/iamaziz/PyDataset



Carga de dados com pydataset

```
import pydataset
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-64-7a2fd7ba3d74> in <module>()
----> 1 import pydataset

ImportError: No module named pydataset
```

Carga de dados com pydataset

```
import pydataset
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-64-7a2fd7ba3d74> in <module>()
----> 1 import pydataset

ImportError: No module named pydataset
```

```
mpp-mac:data_science marciopalheta$ pip install pydataset
Collecting pydataset
  Downloading pydataset-0.2.0.tar.gz (15.9MB)
    0% |                                        | 10kB 2.4MB/s eta
    0% |                                        | 20kB 301kB/s eta
    0% |                                        | 30kB 367kB/s eta
    0% |                                        | 40kB 299kB/s eta
    0% |                                        | 51kB 319kB/s eta
    0% ||                                       | 61kB 383kB/s eta
```

Carga de dados com pydataset

```
import pydataset
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-64-7a2fd7ba3d74> in <module>()
----> 1 import pydataset

ImportError: No module named pydataset
```

```
mpp-mac:data_science marciopalheta$ pip install pydataset
Collecting pydataset
  Downloading pydataset-0.2.0.tar.gz (15.9MB)
    0% |                                        | 10kB 2.4MB/s eta
    0% |                                        | 20kB 301kB/s eta
    0% |                                        | 30kB 367kB/s eta
    0% |                                        | 40kB 299kB/s eta
    0% |                                        | 51kB 319kB/s eta
    0% ||                                       | 61kB 383kB/s eta
```

Exercício 22: Lista de 757 bases

```
import pydataset

# Exibir lista dos 757 datasets disponíveis
pydataset.data()
```

	dataset_id	title
0	AirPassengers	Monthly Airline Passenger Numbers 1949-1960
1	BJsales	Sales Data with Leading Indicator
2	BOD	Biochemical Oxygen Demand
3	Formaldehyde	Determination of Formaldehyde
4	HairEyeColor	Hair and Eye Color of Statistics Students

Exercício 23: dados do titanic

```
# Pegar o dataset do Tintanic
titanic = pydataset.data('titanic')
# Exibir os primeiros 5
titanic.head()
```

	class	age	sex	survived
1	1st class	adults	man	yes
2	1st class	adults	man	yes
3	1st class	adults	man	yes
4	1st class	adults	man	yes
5	1st class	adults	man	yes

Exercício 24: últimas instâncias

```
# Pegar as 5 últimas instâncias
titanic.tail()
```

	class	age	sex	survived
1312	3rd class	child	women	no
1313	3rd class	child	women	no
1314	3rd class	child	women	no
1315	3rd class	child	women	no
1316	3rd class	child	women	no

Exercício 25: descrição geral

```
# Descrição do dataset
titanic.describe()
```

	class	age	sex	survived
count	1316	1316	1316	1316
unique	3	2	2	2
top	3rd class	adults	man	no
freq	706	1207	869	817

Exercício 26: value_counts

```
# Detalhes da classe
titanic['class'].value_counts()
```

```
3rd class    706
1st class    325
2nd class    285
Name: class, dtype: int64
```

```
# Detalhes da idade
titanic['age'].value_counts()
```

```
adults     1207
child       109
Name: age, dtype: int64
```

Acesso a banco de dados

- A biblioteca db.py oferece um conjunto de ferramentas que facilita a interação com Bancos de Dados Relacionais.
- https://github.com/yhat/db.py
- $ pip install db.py
- PostgreSQL, MySQL, SQLite, Redshift, MS SQL Server, Oracle

Exercício 27: DemoDB

Acesso a banco de dados - db.py

```
import pandas as pd
# Importa o banco de dados de demo
from db import DemoDB

# Carregar o banco de dados
database = DemoDB()
```

```
Indexing schema. This will take a second...finished!
```

```
# Exibir as tabelas
database.tables
```

```
Refreshing schema. Please wait...done!
```

Exercício 28: Tabelas

```
# Exibir as tabelas
database.tables
```

```
Refreshing schema. Please wait...done!
```

Schema	Table	Columns
public	Album	AlbumId, Title, ArtistId
public	Artist	ArtistId, Name
public	Customer	CustomerId, FirstName, LastName, Company, Address, City, State, Country, PostalCode, Phone, Fax, Email, SupportRepId
public	Employee	EmployeeId, LastName, FirstName, Title, ReportsTo, BirthDate, HireDate, Address, City, State, Country, PostalCode, Phone, Fax, Email
public	Genre	GenreId, Name
public	Invoice	InvoiceId, CustomerId, InvoiceDate, BillingAddress, BillingCity, BillingState, BillingCountry, BillingPostalCode, Total
public	InvoiceLine	InvoiceLineId, InvoiceId, TrackId, UnitPrice, Quantity
public	MediaType	MediaTypeId, Name
public	Playlist	PlaylistId, Name
public	PlaylistTrack	PlaylistId, TrackId
public	Track	TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, Milliseconds, Bytes, UnitPrice

Exercício 29: Customer

```
# Carrgando uma referência para a tabela de CLIENTES
clientes = database.tables.Customer
print 'type(clientes)', type(clientes)
clientes
```

```
type(clientes) <class 'db.table.Table'>
```

Column	Type	Foreign Keys	Reference Keys
CustomerId	INTEGER		Invoice.CustomerId
FirstName	NVARCHAR(40)		
LastName	NVARCHAR(20)		
Company	NVARCHAR(80)		
Address	NVARCHAR(70)		
City	NVARCHAR(40)		
State	NVARCHAR(40)		
Country	NVARCHAR(40)		
PostalCode	NVARCHAR(10)		
Phone	NVARCHAR(24)		
Fax	NVARCHAR(24)		
Email	NVARCHAR(60)		
SupportRepId	INTEGER	Employee.EmployeeId	

Exercício 30: Customer.all

```
# Carregar o DataFrame com TODOS os dados de clientes
clientes = database.tables.Customer.all()
print 'type(clientes)', type(clientes)
clientes
```

```
type(clientes) <class 'pandas.core.frame.DataFrame'>
```

	CustomerId	FirstName	LastName	Company	Address	City	State
0	1	Luís	Gonçalves	Embraer - Empresa Brasileira de Aeronáutica S.A.	Av. Brigadeiro Faria Lima, 2170	São José dos Campos	SP
1	2	Leonie	Köhler	None	Theodor-Heuss-Straße 34	Stuttgart	None
2	3	François	Tremblay	None	1498 rue Bélanger	Montréal	QC
3	4	Bjørn	Hansen	None	Ullevålsveien 14	Oslo	None

Exercício 31: database.query

```
# Carregar o DataFrame com seleção de registros
clientes_query = database.query('Select * from Customer where CustomerId < 5')
print 'type(clientes_query)', type(clientes_query)
clientes_query
```

```
type(clientes_query) <class 'pandas.core.frame.DataFrame'>
```

	CustomerId	FirstName	LastName	Company	Address	City	State	Country	PostalCode
0	1	Luís	Gonçalves	Embraer - Empresa Brasileira de Aeronáutica S.A.	Av. Brigadeiro Faria Lima, 2170	São José dos Campos	SP	Brazil	12227-000
1	2	Leonie	Köhler	None	Theodor-Heuss-Straße 34	Stuttgart	None	Germany	70174
2	3	François	Tremblay	None	1498 rue Bélanger	Montréal	QC	Canada	H2G 1A7
3	4	Bjørn	Hansen	None	Ullevålsveien 14	Oslo	None	Norway	0171

DataFrame.iloc

- Indexador Pandas usado para localização de instâncias por índices
- Sintaxe geral:
  - data.iloc[<row selection>, <column selection>]
- Com DataFrame.iloc, linhas e colunas podem ser acessadas como vetores, iniciando na posição ZERO

DataFrame de estados

Filtros usando DataFrame.iloc

```
# Apenas os 5 primeiros estados
estados.head()
```

	Estado	Capital
Sigla		
AC	Acre	Rio Branco
AL	Alagoas	Maceió
AP	Amapá	Macapá
AM	Amazonas	Manaus
BA	Bahia	Salvador

Exercício 32: iloc[0]

```
# Trabalhando com posição
# Devolve os dados do 1o estado da coleção
estados.iloc[0]
```

```
Estado            Acre
Capital     Rio Branco
Name: AC, dtype: object
```

Exercício 33: slicing

In [274]:
```
# Filtro de estados usando slicing
# Retorna da posição 1 à posição 15
# Pulando de 2 em 2
estados.iloc[1:15:2]
```

Out[274]:

	Estado	Capital
Sigla		
AL	Alagoas	Maceió
AM	Amazonas	Manaus
CE	Ceara	Fortaleza
ES	Espírito Santo	Vitória
MA	Maranhão	São Luiz
MS	Mato Grosso do Sul	Campo Grande
PA	Pará	Belém

Exercício 34: [:10]

```
# Retorna as 10 primeiras linhas
censo_amazonas.iloc[:10]
```

	Código	Municípios	Gentílico	População 2010	Área territorial 2015 (km²)
0	1300029	Alvarães	alvarãense	14088	5923.46
1	1300060	Amaturá	amaturaense	9467	4754.11
2	1300086	Anamã	anamãense	10214	2453.94
3	1300102	Anori	anoriense	16317	6036.36
4	1300144	Apuí	apuiense	18007	54244.92
5	1300201	Atalaia do Norte	atalaiense	15153	76345.16
6	1300300	Autazes	autazense	32135	7623.27
7	1300409	Barcelos	barcelense	25718	122450.77
8	1300508	Barreirinha	barreirinhense	27355	5750.56
9	1300607	Benjamin Constant	benjamin-constantense	33411	8785.32

Exercício 35: [:5,1]

```
# Retornas as 5 primeiras linhas
# Exibindo apenas a coluna 1
censo_amazonas.iloc[:5,1]
```

```
0     Alvarães
1      Amaturá
2        Anamã
3        Anori
4         Apuí
Name: Municípios, dtype: object
```

Exercício 36: [:5, 1:4]

```
# Retornas as 5 primeiras linhas
# Exibido as colunas de 1 a 3
censo_amazonas.iloc[:5, 1:4]
```

	Municípios	Gentílico	População 2010
0	Alvarães	alvarãense	14088
1	Amaturá	amaturaense	9467
2	Anamã	anamãense	10214
3	Anori	anoriense	16317
4	Apuí	apuiense	18007

Exercício 37: linhas fixas

In [270]:
```
# Retornas as linhas 1, 3, 4 e 37
# Exibido as colunas de 1 a 3
censo_amazonas.iloc[[1,3, 4, 37], 1:4]
```

Out[270]:

	Municípios	Gentílico	População 2010
1	Amaturá	amaturaense	9467
3	Anori	anoriense	16317
4	Apuí	apuiense	18007
37	Manaus	manauara	1802014

Exercício 38: coordenadas fixas

```
# Retornas as linhas 1, 3, 4 e 37
# Exibido as colunas 1, 3 e 6
censo_amazonas.iloc[[1,3, 4, 37], [1, 3, 6]]
```

	Municípios	População 2010	Série revisada
1	Amaturá	9467	62093
3	Anori	16317	166040
4	Apuí	18007	183915
37	Manaus	1802014	53305515

DataFrame.loc

- A DataFrame.loc é usada para seleção de instâncias baseada em labels
- Dados.loc[2]: retorna a linha de índice 2
- Dados.loc[‘matemática’]: retorna a linha de índice ‘matemática’
- Dados.loc[[‘seg’, ‘sex’]]: retorna as linhas de índices ‘seg’ e ‘sex’

Exercício 39: filtros por índices

```
In [196]: # Seleção de um índice
          estados.loc['AM']
```

```
Out[196]: Estado      Amazonas
          Capital       Manaus
          Name: AM, dtype: object
```

```
In [202]: # Seleção por índices
          estados.loc[['AM', 'PA', 'MA']]
```

Out[202]:

	Estado	Capital
Sigla		
AM	Amazonas	Manaus
PA	Pará	Belém
MA	Maranhão	São Luiz

Exercício 40: slicing de índices

```
# Seleção por slicing de índices
# Só podemos usar quando o índice for numérico
copacabana.loc[2:20:3]
```

	Posicao	Quartos	Vagas	DistIpanema	DistPraia	DistFavela	RendaMedia	RendaMovel
2	0	2.00	0.01	2448	772	229	1803724	1512475
5	0	2.00	1.00	2358	586	287	1165764	1177933
8	0	1.00	0.01	555	332	306	1140573	1064998
11	1	0.01	0.01	555	332	306	1140573	1064998
14	1	2.00	0.01	2498	716	148	1281444	1281444
17	0	2.00	0.01	2835	330	629	1124331	727350
20	1	0.01	0.01	2057	307	621	1395924	1176888

Exercício 41: slicing e colunas

```
# Seleção por slicing de índices, exibindo 1 coluna
copacabana.loc[2:5]['Quartos']
```

```
2    2.0
3    2.0
4    2.0
5    2.0
Name: Quartos, dtype: float64
```

```
# Seleção por slicing de índices, exibindo várias colunas
copacabana.loc[2:5][['Quartos', 'Vagas', 'Idade']]
```

	Quartos	Vagas	Idade
2	2.0	0.01	44.0
3	2.0	0.01	43.0
4	2.0	1.00	42.0
5	2.0	1.00	42.0

Exercício 42: slicing com strings

```
# Seleção por slicing de índices e colunas
copacabana.loc[2:5, 'Posicao':'DistPraia']
```

	Posicao	Quartos	Vagas	DistIpanema	DistPraia
2	0	2.0	0.01	2448	772
3	0	2.0	0.01	1615	428
4	0	2.0	1.00	2358	586
5	0	2.0	1.00	2358	586

Gerando séries booleanas

- Em pandas, podemos utilizar expressões booleanas para filtrar instâncias
- O comando: copacabana['Quartos'] > 5 devolve uma série que indica se cada instância do DataFrame possui (True) ou não(False) mais de 5 quartos

Exercício 43: séries booleanas

```
print 'type(copacabana):\ n', type(copacabana)

# Aplicando filtros a uma coluna do DataFrame
filtro = copacabana['Quartos'] > 5
print 'type(filtro)', type(filtro)

filtro_nro_quartos
```

```
type(copacabana):\ n <class 'pandas.core.frame.DataFrame'>
type(filtro) <class 'pandas.core.series.Series'>

0       False
1       False
2       False
3       False
4       False
```

Filtro com séries booleanas

- Outro formato de filtro aceito pela função DataFrame.loc é baseado em séries booleanas
- Neste caso, DataFrame.loc assume que a série indica quais posições devem (True) ou não(False) devem ser retornadas

Exercício 44: value_counts()

```
# nro de quartos e suas frequências
print copacabana['Quartos'].value_counts()
# Deve exibir apenas imóveis que possuam 6 quartos
serie = copacabana['Quartos'] == 6
# Filtro usando uma série booleana
copacabana.loc[serie]
```

```
1.00    477
3.00    439
2.00    422
0.01    258
4.00     76
5.00      2
6.00      1
Name: Quartos, dtype: int64
```

	Posicao	Quartos	Vagas	DistIpanema	DistPraia	DistFavela	RendaMedia
748	1	6.0	2.0	2500	35	743	1524600

Exercício 45: filtro

```
print copacabana['Quartos'].value_counts()
# Imóveis com 5 quartos ou mais
copacabana.loc[copacabana['Quartos'] >= 5]
```

```
1.00    477
3.00    439
2.00    422
0.01    258
4.00     76
5.00      2
6.00      1
Name: Quartos, dtype: int64
```

	Posicao	Quartos	Vagas	DistIpanema	DistPraia	DistFavela	RendaMedia	RendaMovel
173	1	5.0	0.01	2696	154	712	1083455	649733
748	1	6.0	2.00	2500	35	743	1524600	1275377
1521	1	5.0	3.00	3044	31	609	1524600	1168222

Campos calculados

- Na teoria de banco de dados, podemos criar campos calculados, que são campos criados a partir de de associações de outros campos
- Em pandas, podemos gerar séries com resultado de operações e anexa-las a um DataFrame

Exercício 46: cáclulo

```
# Criação de uma série com o resultado da multiplicação
resultado = copacabana['AreaConstruida'] * copacabana['VAL_UNIT']
print 'type(resultado)', type(resultado)
resultado
```

```
type(resultado) <class 'pandas.core.series.Series'>

0         416005
1         460009
2         720012
3         990000
4         909976
5         790024
6         670015
7         249990
8         319990
```

Exercício 47: df['col'].describe

```
nova_coluna = copacabana['AreaConstruida'] * copacabana['VAL_UNIT']
# Criação da coluna Total
copacabana['TOTAL'] = nova_coluna

# Acesso à descrição da nova coluna
copacabana['TOTAL'].describe()
```

```
count    1.675000e+03
mean     5.862613e+05
std      4.804788e+05
min      6.000000e+04
25%      3.000000e+05
50%      4.500320e+05
75%      7.200115e+05
max      6.199912e+06
Name: TOTAL, dtype: float64
```

Exercício 48: Seleção de colunas

```
# Seleção de colunas a exibir, combinada com slicing
copacabana[['AreaConstruida', 'VAL_UNIT', 'TOTAL']][2:20:3]
```

	AreaConstruida	VAL_UNIT	TOTAL
2	58	12414	720012
5	68	11618	790024
8	22	14545	319990
11	22	13636	299992
14	62	10081	625022
17	64	5313	340032

Dados categóricos

- Em ciência de dados, é comum o uso de variáveis qualitativas e quantitativas (contínuas ou discretas)
- Variáveis contínuas categorizadas:
  - Idade: faixas etárias
  - Resultado do exame: normal/anormal
  - Peso: < 60, [60,100), [100,150) e >=250

Exercício 49: carga do titanic

Dados categóricos

In [301]:
```
import pandas as pd
import pydataset

# carregando dados do tintanic
titanic = pydataset.data('titanic')
# exibindo as 5 primeiras linhas
titanic.head()
```

Out[301]:

	class	age	sex	survived
1	1st class	adults	man	yes
2	1st class	adults	man	yes
3	1st class	adults	man	yes
4	1st class	adults	man	yes
5	1st class	adults	man	yes

Exercício 50: análise 'class'

```
In [530]: # lista de colunas
          titanic.columns

Out[530]: Index([u'class', u'age', u'sex', u'survived'], dtype='object')

In [531]: #Detalhes da 'class' - apenas 3 valores únicos
          titanic['class'].describe()

Out[531]: count          1316
          unique            3
          top       3rd class
          freq            706
          Name: class, dtype: object

In [532]: # memória usada para a coluna 'class'
          titanic['class'].nbytes

Out[532]: 10528
```

Exercício 51: performance

```
# memória usada para a coluna 'class'
titanic['class'].nbytes
```

```
10528
```

```
%%time
# %%time - calcular o tempo de resposta

# execucao da consulta
titanic['class'][:5] == '3rd class'
```

```
CPU times: user 566 µs, sys: 92 µs, total: 658 µs
Wall time: 604 µs

1    False
2    False
3    False
4    False
5    False
Name: class, dtype: bool
```

Exercício 52: category

```
# Usando o tipo interno do pandas
titanic['class'] = titanic['class'].astype('category')
titanic['class'].describe()
```

```
count            1316
unique              3
top         3rd class
freq              706
Name: class, dtype: object
```

Exercício 53: performance

```
# Uso de memória cai de 10528 para 1340
titanic['class'].nbytes
```

```
1340
```

```
%%time
# Wall time diminuído de 604 para 516
titanic['class'][:5] == '3rd class'
```

```
CPU times: user 493 µs, sys: 55 µs, total: 548 µs
Wall time: 516 µs

1     False
2     False
3     False
4     False
5     False
Name: class, dtype: bool
```

Data missing

- Data missing (ou Dados faltantes) ocorre quando não temos todos os dados para uma determinada instância
- Podemos escolher não lidar com dados faltantes
- Ou definir valores padrão a serem aplicados aos atributos não informados

Exercício 54: Dados faltantes

```
import numpy as np
import pydataset

#Criação de um dicionário com dados de alunos
alunos_dic = {
    'nome': ['João', 'Maria', 'José', np.nan, 'Pedro', 'Judas', 'Tiago'],
    'sexo': ['M', 'F', 'M', np.nan, 'M', 'M', np.nan],
    'idade': [14, 13, np.nan, np.nan, 15, 13, 14],
    'nota': [4, 10, 7, np.nan, 8, 9, 7]
}

# conversão do dicionário para DataFrame
alunos = pd.DataFrame(alunos_dic)
alunos
```

	idade	nome	nota	sexo
0	14.0	João	4.0	M
1	13.0	Maria	10.0	F
2	NaN	José	7.0	M
3	NaN	NaN	NaN	NaN
4	15.0	Pedro	8.0	M
5	13.0	Judas	9.0	M
6	14.0	Tiago	7.0	NaN

Exercício 55: sem dados

```
# Gerar novo DataFrame
# Remove todas as linhas que tenham
# pelo menos um NaN

alunos.dropna()
```

	idade	nome	nota	sexo
0	14.0	João	4.0	M
1	13.0	Maria	10.0	F
4	15.0	Pedro	8.0	M
5	13.0	Judas	9.0	M

Exercício 56: how='all'

```
In [544]: # Remove todas as linhas em que
          # todos os atributos forem NaN

          alunos.dropna(how='all')
```

Out[544]:

	idade	nome	nota	sexo
0	14.0	João	4.0	M
1	13.0	Maria	10.0	F
2	NaN	José	7.0	M
4	15.0	Pedro	8.0	M
5	13.0	Judas	9.0	M
6	14.0	Tiago	7.0	NaN

Exercício 57: nova coluna

In [545]:
```
# Vamos incluir uma nova coluna
alunos['serie'] = np.nan
alunos
```

Out[545]:

	idade	nome	nota	sexo	serie
0	14.0	João	4.0	M	NaN
1	13.0	Maria	10.0	F	NaN
2	NaN	José	7.0	M	NaN
3	NaN	NaN	NaN	NaN	NaN
4	15.0	Pedro	8.0	M	NaN
5	13.0	Judas	9.0	M	NaN
6	14.0	Tiago	7.0	NaN	NaN

Exercício 58: axis=1

In [548]:
```
# Remove todas as COLUNAS (axis=1)
# em que todos os valores forem NaN

alunos.dropna(how='all', axis=1)
```

Out[548]:

	idade	nome	nota	sexo
0	14.0	João	4.0	M
1	13.0	Maria	10.0	F
2	NaN	José	7.0	M
3	NaN	NaN	NaN	NaN
4	15.0	Pedro	8.0	M
5	13.0	Judas	9.0	M
6	14.0	Tiago	7.0	NaN

Exercício 59: thresh=3

In [550]:
```
# Remove todas as LINHAS (axis=0)
# em que PELO MENOS 3 valores sejam NaN

alunos.dropna(thresh=3)
```

Out[550]:

	idade	nome	nota	sexo	serie
0	14.0	João	4.0	M	NaN
1	13.0	Maria	10.0	F	NaN
2	NaN	José	7.0	M	NaN
4	15.0	Pedro	8.0	M	NaN
5	13.0	Judas	9.0	M	NaN
6	14.0	Tiago	7.0	NaN	NaN

Exercício 60: fillna(0)

In [552]:
```
# Preencher todos os valores
# NaN com zero
alunos.fillna(0)
```

Out[552]:

	idade	nome	nota	sexo	serie
0	14.0	João	4.0	M	0.0
1	13.0	Maria	10.0	F	0.0
2	0.0	José	7.0	M	0.0
3	0.0	0	0.0	0	0.0
4	15.0	Pedro	8.0	M	0.0
5	13.0	Judas	9.0	M	0.0
6	14.0	Tiago	7.0	0	0.0

Exercício 61: serie sem NaN

```
In [554]: # Gerar uma série de nomes.
          # Preencher NaN com 'TOMÉ'
          alunos['nome'].fillna('TOMÉ')

Out[554]: 0      João
          1     Maria
          2      José
          3      TOMÉ
          4     Pedro
          5     Judas
          6     Tiago
          Name: nome, dtype: object
```

Exercício 62: inplace=True

In [557]:
```
# Alterar o DataFrame
# Preencher NaN com 'TOMÉ'
alunos['nome'].fillna('TOMÉ', inplace=True)
alunos
```

Out[557]:

	idade	nome	nota	sexo	serie
0	14.0	João	4.0	M	NaN
1	13.0	Maria	10.0	F	NaN
2	NaN	José	7.0	M	NaN
3	NaN	TOMÉ	NaN	NaN	NaN
4	15.0	Pedro	8.0	M	NaN
5	13.0	Judas	9.0	M	NaN
6	14.0	Tiago	7.0	NaN	NaN

Exercício 63: usando a média

```
# Calculando a idade media
alunos['idade'].mean()
```

```
13.800000000000001
```

```
# Atualizando idades NaN com a idade media
alunos['idade'].fillna(alunos['idade'].mean(), inplace=True)
alunos
```

	idade	nome	nota	sexo	serie
0	14.0	João	4.0	M	NaN
1	13.0	Maria	10.0	F	NaN
2	13.8	José	7.0	M	NaN
3	13.8	TOMÉ	NaN	NaN	NaN
4	15.0	Pedro	8.0	M	NaN
5	13.0	Judas	9.0	M	NaN
6	14.0	Tiago	7.0	NaN	NaN

Exercício 64: notnull()

```
# Filtrar instâncias com nota e sexo not null
alunos[alunos['nota'].notnull() & alunos['sexo'].notnull()]
```

	idade	nome	nota	sexo	serie
0	14.0	João	4.0	M	NaN
1	13.0	Maria	10.0	F	NaN
2	13.8	José	7.0	M	NaN
4	15.0	Pedro	8.0	M	NaN
5	13.0	Judas	9.0	M	NaN

Agregação e Agrupamento

- O pandas permite que realizemos agrupamentos, em torno de termos do DataFrame. Usamos o groupby()
- Aggregate: Permite que agreguemos informações a cerca de um determinado agrupamento

Exercício 65: eleições

Aggregate e Grouping no DataFrame

```
import pandas as pd
import numpy as np

# Dados das eleições primárias dos EUA
eleicoes = pd.read_csv('primary-results.csv')
eleicoes.head()
```

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes
0	Alabama	AL	Autauga	1001.0	Democrat	Bernie Sanders	544	0.182
1	Alabama	AL	Autauga	1001.0	Democrat	Hillary Clinton	2387	0.800
2	Alabama	AL	Baldwin	1003.0	Democrat	Bernie Sanders	2694	0.329
3	Alabama	AL	Baldwin	1003.0	Democrat	Hillary Clinton	5290	0.647
4	Alabama	AL	Barbour	1005.0	Democrat	Bernie Sanders	222	0.078

```
len(eleicoes)
```

```
24611
```

Exercício 66: eleições

```
# No aggregate, analisamos a coluna 'votes', aplicando 3 funções
eleicoes.groupby('candidate').aggregate({'votes':[min, np.mean, max]})
```

	votes		
	min	mean	max
candidate			
No Preference	0	23.225071	580
Uncommitted	0	0.434343	16
Ben Carson	0	338.258238	9945
Bernie Sanders	0	2844.019501	434656
Carly Fiorina	0	139.366972	3612
Chris Christie	1	223.422018	7144
Donald Trump	0	3709.576408	179130

Exercício 67: mais votos

```
# Onde Hillary Clinton teve 590502 ?
eleicoes[eleicoes['votes']==590502]
```

	state	state_abbreviation	county	fips	party	candidate
1386	California	CA	Los Angeles	6037.0	Democrat	Hillary Clinton

Exercício 68: votos e percentual

```
eleicoes.groupby('candidate').aggregate({'votes':[min, np.mean, max],
                                         'fraction_votes':[min, np.mean, max]})
```

	votes			fraction_votes		
	min	mean	max	min	mean	max
candidate						
No Preference	0	23.225071	580	0.000	0.006484	0.030000
Uncommitted	0	0.434343	16	0.000	0.000455	0.013000
Ben Carson	0	338.258238	9945	0.000	0.058941	0.415000
Bernie Sanders	0	2844.019501	434656	0.000	0.493316	1.000000
Carly Fiorina	0	139.366972	3612	0.000	0.022097	0.117000
Chris Christie	1	223.422018	7144	0.002	0.017773	0.087195
Donald Trump	0	3709.576408	179130	0.000	0.466217	0.915000
Hillary Clinton	0	3731.855410	590502	0.000	0.461302	1.000000

Exercício 69: 100%

```
# Em quais distritos tivemos candidatos com 100% de votos?
eleicoes[eleicoes['fraction_votes'] == 1]
```

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes
475	Alaska	AK	State House District 12	90200112.0	Democrat	Bernie Sanders	10	1.0
499	Alaska	AK	State House District 23	90200123.0	Democrat	Bernie Sanders	13	1.0
513	Alaska	AK	State House District 3	90200103.0	Democrat	Bernie Sanders	7	1.0
517	Alaska	AK	State House District 31	90200131.0	Democrat	Bernie Sanders	16	1.0
519	Alaska	AK	State House District 32	90200132.0	Democrat	Bernie Sanders	13	1.0
531	Alaska	AK	State House District 38	90200138.0	Democrat	Bernie Sanders	17	1.0
537	Alaska	AK	State House District 40	90200140.0	Democrat	Bernie Sanders	12	1.0
539	Alaska	AK	State House District 5	90200105.0	Democrat	Bernie Sanders	15	1.0
541	Alaska	AK	State House District 6	90200106.0	Democrat	Bernie Sanders	12	1.0
545	Alaska	AK	State House District 8	90200108.0	Democrat	Bernie Sanders	7	1.0
4198	Idaho	ID	Clark	16033.0	Democrat	Bernie Sanders	4	1.0
8121	Maine	ME	Abbot	92300001.0	Democrat	Bernie Sanders	1	1.0

Exercício 70: Hillary Clinton 100%

```
# Em quais distritos Hillary Clinton teve 100% dos votos?

eleicoes[(eleicoes['fraction_votes']==1) & (eleicoes['candidate']=='Hillary Clinton')][-10:]
```

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes
8770	Maine	ME	Otis	92300338.0	Democrat	Hillary Clinton	1	1.0
8828	Maine	ME	Portage Lake	92300364.0	Democrat	Hillary Clinton	1	1.0
8866	Maine	ME	Roque Bluffs	92300384.0	Democrat	Hillary Clinton	1	1.0
8932	Maine	ME	Springfield	92300421.0	Democrat	Hillary Clinton	1	1.0
8974	Maine	ME	Sweden	92300438.0	Democrat	Hillary Clinton	1	1.0
9018	Maine	ME	Wade	92300464.0	Democrat	Hillary Clinton	1	1.0
9052	Maine	ME	Wellington	92300480.0	Democrat	Hillary Clinton	1	1.0
9070	Maine	ME	Westmanland	92300488.0	Democrat	Hillary Clinton	1	1.0
14088	Nebraska	NE	Hayes	31085.0	Democrat	Hillary Clinton	9	1.0
14174	Nebraska	NE	Thomas	31171.0	Democrat	Hillary Clinton	1	1.0

Exercício 71: filter()

```
# Filtrar distritos de estados com mais de 1.000.000 de votos
def votes_filter(x):
    return x['votes'].sum() > 1000000
eleicoes.groupby('state').filter(votes_filter)
```

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes
0	Alabama	AL	Autauga	1001.0	Democrat	Bernie Sanders	544	0.182
1	Alabama	AL	Autauga	1001.0	Democrat	Hillary Clinton	2387	0.800
2	Alabama	AL	Baldwin	1003.0	Democrat	Bernie Sanders	2694	0.329
3	Alabama	AL	Baldwin	1003.0	Democrat	Hillary Clinton	5290	0.647
4	Alabama	AL	Barbour	1005.0	Democrat	Bernie Sanders	222	0.078
5	Alabama	AL	Barbour	1005.0	Democrat	Hillary Clinton	2567	0.906
6	Alabama	AL	Bibb	1007.0	Democrat	Bernie Sanders	246	0.197
7	Alabama	AL	Bibb	1007.0	Democrat	Hillary Clinton	942	0.755
8	Alabama	AL	Blount	1009.0	Democrat	Bernie Sanders	395	0.386
9	Alabama	AL	Blount	1009.0	Democrat	Hillary Clinton	564	0.551
10	Alabama	AL	Bullock	1011.0	Democrat	Bernie Sanders	178	0.066

Exercício 72: estatísticas

- Quantos votos foram apurado no estado do Alabama?
- Quantos votos Hillary Clinton recebeu no estado do Wisconsin?
- Qual a votação de cada candidato, por estado?

Exercício 72: estatísticas

```
# Quantos votos foram apurado no estado do Alabama ?
alabama = eleicoes[eleicoes['state_abbreviation']=='AL']
alabama['votes'].sum()
```

```
1223959
```

```
# Quantos votos Hillary Clinton recebeu no estado do Wisconsin ?
eleicoes[(eleicoes['state_abbreviation']=='WI') &
         (eleicoes['candidate']=='Hillary Clinton')]['votes'].sum()
```

```
432767
```

```
# Qual a votação de cada candidato, por estado ?
eleicoes.groupby(['state_abbreviation', 'candidate'])
eleicoes.groupby(['state_abbreviation', 'candidate']).aggregate({'votes':[sum], 'fraction_votes':[sum]})
```

		votes	fraction_votes
		sum	sum
state_abbreviation	candidate		
AK	Ben Carson	2401	4.442
	Bernie Sanders	440	32.941
	Donald Trump	7346	13.281
	Hillary Clinton	99	7.059
	John Kasich	892	1.862
	Marco Rubio	3318	6.409
	Ted Cruz	7973	14.010
	Ben Carson	87517	6.814

Merge e Join com DataFrame

- Imagine que precisamos complementar um DataFrame com dados de outro
- Em banco de dados faríamos junção das tabelas usando alguns “joins”
- Em Pandas, podemos fazer merge dos DataFrames, a partir de colunas com o mesmo nome

Exercício 73: DemoDB

Merge (join) de DataFrames

```
import pandas as pd
from db import DemoDB

database = DemoDB()
```

```
Indexing schema. This will take a second...finished!
```

```
database.tables
```

```
Refreshing schema. Please wait...done!
```

Exercício 74: DemoDB

```
database.tables
```

```
Refreshing schema. Please wait...done!
```

Schema	Table	Columns
public	Album	AlbumId, Title, ArtistId
public	Artist	ArtistId, Name
public	Customer	CustomerId, FirstName, LastName, Company, Address, City, State, Country, PostalCode, Phone, Fax, Email, SupportRepId
public	Employee	EmployeeId, LastName, FirstName, Title, ReportsTo, BirthDate, HireDate, Address, City, State, Country, PostalCode, Phone, Fax, Email
public	Genre	GenreId, Name
public	Invoice	InvoiceId, CustomerId, InvoiceDate, BillingAddress, BillingCity, BillingState, BillingCountry, BillingPostalCode, Total
public	InvoiceLine	InvoiceLineId, InvoiceId, TrackId, UnitPrice, Quantity
public	MediaType	MediaTypeId, Name
public	Playlist	PlaylistId, Name
public	PlaylistTrack	PlaylistId, TrackId
public	Track	TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, Milliseconds, Bytes, UnitPrice

Exercício 75: Album.all()

```
# Carregando o DataFrame com Albuns
album = database.tables.Album.all()
album.head()
```

	AlbumId	Title	ArtistId
0	1	For Those About To Rock We Salute You	1
1	2	Balls to the Wall	2
2	3	Restless and Wild	2
3	4	Let There Be Rock	1
4	5	Big Ones	3

Exercício 76: Artist.all()

```
# Carregando o DataFrame com dados de Artistas
artista = database.tables.Artist.all()
artista.head()
```

	ArtistId	Name
0	1	AC/DC
1	2	Accept
2	3	Aerosmith
3	4	Alanis Morissette
4	5	Alice In Chains

Exercício 77: pd.merge()

```
# Merge dos DataFrames de Album e Artista
artista_album = pd.merge(artista, album)
artista_album.head()
```

	ArtistId	Name	AlbumId	Title
0	1	AC/DC	1	For Those About To Rock We Salute You
1	1	AC/DC	4	Let There Be Rock
2	2	Accept	2	Balls to the Wall
3	2	Accept	3	Restless and Wild
4	3	Aerosmith	5	Big Ones

Exercício 78: on='ArtistId'

```
# Merge dos DataFrames de Album e Artista, especificando a coluna
artista_album = pd.merge(artista, album, on='ArtistId')
artista_album.head()
```

	ArtistId	Name	AlbumId	Title
0	1	AC/DC	1	For Those About To Rock We Salute You
1	1	AC/DC	4	Let There Be Rock
2	2	Accept	2	Balls to the Wall
3	2	Accept	3	Restless and Wild
4	3	Aerosmith	5	Big Ones

Exercício 79: rename

```
# Renomeando colunas
album.rename(columns={'ArtistId':'Artist_Id'}, inplace=True)
album.head()
```

	AlbumId	Title	Artist_Id
0	1	For Those About To Rock We Salute You	1
1	2	Balls to the Wall	2
2	3	Restless and Wild	2
3	4	Let There Be Rock	1
4	5	Big Ones	3

Exercício 80: left_on, right_on

```
# Merge dos DataFrames de Album e Artista, com colunas diferentes
artista_album = pd.merge(artista, album, left_on='ArtistId', right_on='Artist_Id')
artista_album.head()
```

	ArtistId	Name	AlbumId	Title	Artist_Id
0	1	AC/DC	1	For Those About To Rock We Salute You	1
1	1	AC/DC	4	Let There Be Rock	1
2	2	Accept	2	Balls to the Wall	2
3	2	Accept	3	Restless and Wild	2
4	3	Aerosmith	5	Big Ones	3

Exercício 81: drop

```
# Merge dos DataFrames de Album e Artista, com colunas diferentes,
# sem campos duplicados
artista_album = pd.merge(artista, album, left_on='ArtistId',
                         right_on='Artist_Id').drop('Artist_Id', axis=1)
artista_album.head()
```

	ArtistId	Name	AlbumId	Title
0	1	AC/DC	1	For Those About To Rock We Salute You
1	1	AC/DC	4	Let There Be Rock
2	2	Accept	2	Balls to the Wall
3	2	Accept	3	Restless and Wild
4	3	Aerosmith	5	Big Ones

Exercício 82: notas e codigos

In [695]:
```
notas = pd.DataFrame(
    {
        'nome': ['Maria', 'Sofia'],
        'nota': [8, 9],
    }
)
notas
```

Out[695]:

	nome	nota
0	Maria	8
1	Sofia	9

In [696]:
```
codigos = pd.DataFrame(
    {
        'nome': ['João', 'Pedro', 'Maria'],
        'cod': [1, 2, 3],
    }
)
codigos
```

Out[696]:

	cod	nome
0	1	João
1	2	Pedro
2	3	Maria

Exercício 83: inner

```
# Merge de códigos e notas, pelo nome
codigos_notas = pd.merge(codigos, notas)
codigos_notas
```

	cod	nome	nota
0	3	Maria	8

Exercício 84: how=outer

```
# Merge de códigos e notas, pelo nome
# how='outer': traz todas as linhas
pd.merge(codigos, notas, how='outer')
```

	cod	nome	nota
0	1.0	João	NaN
1	2.0	Pedro	NaN
2	3.0	Maria	8.0
3	NaN	Sofia	9.0

Exercício 85: how=left

```
# Merge de códigos e notas, pelo nome
# how='left': traz todos os codigos
pd.merge(codigos, notas, how='left')
```

	cod	nome	nota
0	1	João	NaN
1	2	Pedro	NaN
2	3	Maria	8.0

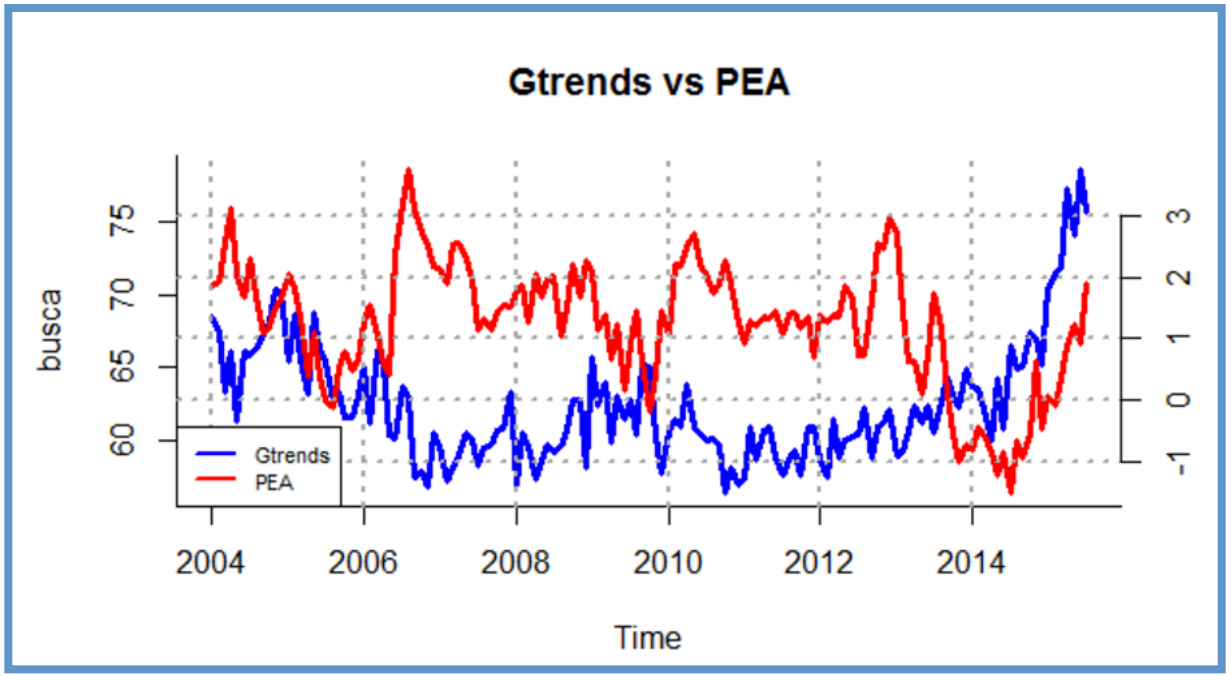
Exercício 86: how=right

```
# Merge de códigos e notas, pelo nome
# how='right': traz todas as notas
pd.merge(codigos, notas, how='right')
```

	cod	nome	nota
0	3.0	Maria	8
1	NaN	Sofia	9

Séries temporais

- Uma série temporal é um conjunto de observações ordenadas no tempo



Exercício 87: rotina.csv

Séries temporais

```
import pandas as pd

#Carga de um dataset com registro de 11 dias de atividades de um indivíduo
rotina = pd.read_csv('rotina.csv', delimiter=';')
rotina.head()
```

	Start time	End time	Activity
0	28/11/11 02:27	28/11/11 10:18	Sleeping
1	28/11/11 10:21	28/11/11 10:23	Toileting
2	28/11/11 10:25	28/11/11 10:33	Showering
3	28/11/11 10:34	28/11/11 10:43	Breakfast
4	28/11/11 10:49	28/11/11 10:51	Grooming

Exercício 88: rotina.csv

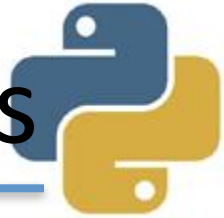
Séries temporais

```
import pandas as pd

#Carga de um dataset com registro de 11 dias de atividades de um indivíduo
rotina = pd.read_csv('rotina.csv', delimiter=';')
print type(rotina)
rotina.head()
```

```
<class 'pandas.core.frame.DataFrame'>
```

	StartTime	EndTime	Activity
0	28/11/2011 02:27	28/11/2011 10:18	Sleeping
1	28/11/2011 10:21	28/11/2011 10:23	Toileting
2	28/11/2011 10:25	28/11/2011 10:33	Showering
3	28/11/2011 10:34	28/11/2011 10:43	Breakfast
4	28/11/2011 10:49	28/11/2011 10:51	Grooming

Exercício 89: columns e dtypes



```
# Exibindo a lista de colunas do DataFrame
rotina.columns
```

```
Index([u'StartTime', u'EndTime', u'Activity'], dtype='object')
```

```
# Exibindo os tipos das colunas
rotina.dtypes
```

```
StartTime     object
EndTime       object
Activity      object
dtype: object
```

Exercício 90: pd.to_datetime

```
# Converter StartTime e EndTime para datetime
rotina['StartTime'] = pd.to_datetime(rotina['StartTime'], format='%d/%m/%Y %H:%M')
rotina['EndTime'] = pd.to_datetime(rotina['EndTime'], format='%d/%m/%Y %H:%M')
rotina.dtypes
```

```
StartTime    datetime64[ns]
EndTime      datetime64[ns]
Activity             object
dtype: object
```

```
# No visual, não percebemos mudanças
rotina.head()
```

	StartTime	EndTime	Activity
0	2011-11-28 02:27:00	2011-11-28 10:18:00	Sleeping
1	2011-11-28 10:21:00	2011-11-28 10:23:00	Toileting
2	2011-11-28 10:25:00	2011-11-28 10:33:00	Showering
3	2011-11-28 10:34:00	2011-11-28 10:43:00	Breakfast
4	2011-11-28 10:49:00	2011-11-28 10:51:00	Grooming

Exercício 91: set_index

```
# O índice da série temporal deve ser o campo data principal
rotina.set_index('StartTime', inplace=True)
rotina.head()
```

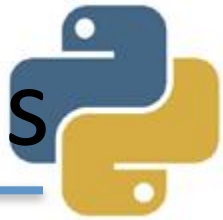
	EndTime	Activity
StartTime		
2011-11-28 02:27:00	2011-11-28 10:18:00	Sleeping
2011-11-28 10:21:00	2011-11-28 10:23:00	Toileting
2011-11-28 10:25:00	2011-11-28 10:33:00	Showering
2011-11-28 10:34:00	2011-11-28 10:43:00	Breakfast
2011-11-28 10:49:00	2011-11-28 10:51:00	Grooming

Exercício 92: filtro por ano

```
# Filtrar as intâncias de 2011
rotina['2011'].head()
```

	EndTime	Activity
StartTime		
2011-11-28 02:27:00	2011-11-28 10:18:00	Sleeping
2011-11-28 10:21:00	2011-11-28 10:23:00	Toileting
2011-11-28 10:25:00	2011-11-28 10:33:00	Showering
2011-11-28 10:34:00	2011-11-28 10:43:00	Breakfast
2011-11-28 10:49:00	2011-11-28 10:51:00	Grooming

Exercício 93: filtro por ano e mês



```
# Filtrar as intâncias de novembro de 2011
rotina['2011-11'].head()
```

	EndTime	Activity
StartTime		
2011-11-28 02:27:00	28/11/2011 10:18	Sleeping
2011-11-28 10:21:00	28/11/2011 10:23	Toileting
2011-11-28 10:25:00	28/11/2011 10:33	Showering
2011-11-28 10:34:00	28/11/2011 10:43	Breakfast
2011-11-28 10:49:00	28/11/2011 10:51	Grooming

Exercício 94: filtro por data

```
In [862]: # Filtrar as intâncias de 28 de novembro de 2011
          rotina['2011-11-28']
```

Out[862]:

	EndTime	Activity
StartTime		
2011-11-28 02:27:00	28/11/2011 10:18	Sleeping
2011-11-28 10:21:00	28/11/2011 10:23	Toileting
2011-11-28 10:25:00	28/11/2011 10:33	Showering
2011-11-28 10:34:00	28/11/2011 10:43	Breakfast
2011-11-28 10:49:00	28/11/2011 10:51	Grooming
2011-11-28 10:51:00	28/11/2011 13:05	Spare_Time/TV
2011-11-28 13:06:00	28/11/2011 13:06	Toileting
2011-11-28 13:09:00	28/11/2011 13:29	Leaving
2011-11-28 13:38:00	28/11/2011 14:21	Spare_Time/TV
2011-11-28 14:22:00	28/11/2011 14:27	Toileting
2011-11-28 14:27:00	28/11/2011 15:04	Lunch
2011-11-28 15:04:00	28/11/2011 15:06	Grooming
2011-11-28 15:07:00	28/11/2011 20:20	Spare_Time/TV
2011-11-28 20:20:00	28/11/2011 20:20	Snack
2011-11-28 20:21:00	29/11/2011 02:06	Spare_Time/TV

Exercício 95: filtro por hora

```
# Filtrar as intâncias de 28 de novembro de 2011
# às 10h
rotina['2011-11-28 10']
```

	EndTime	Activity
StartTime		
2011-11-28 10:21:00	28/11/2011 10:23	Toileting
2011-11-28 10:25:00	28/11/2011 10:33	Showering
2011-11-28 10:34:00	28/11/2011 10:43	Breakfast
2011-11-28 10:49:00	28/11/2011 10:51	Grooming
2011-11-28 10:51:00	28/11/2011 13:05	Spare_Time/TV

Exercício 96: filtro por período

```
rotina['2011-11-28 10:33':'2011-11-28 13:06']
```

	EndTime	Activity
StartTime		
2011-11-28 10:34:00	28/11/2011 10:43	Breakfast
2011-11-28 10:49:00	28/11/2011 10:51	Grooming
2011-11-28 10:51:00	28/11/2011 13:05	Spare_Time/TV
2011-11-28 13:06:00	28/11/2011 13:06	Toileting

Pivot Tables

- Ferramenta que permite a sumarização dos dados de um DataFrame
- Você tem liberdade para definir os critérios da sumarização
- Vamos voltar usar o arquivo das eleições primárias dos EUA

Exercício 97: eleições primárias

Pivot tables

```
import pandas as pd
import numpy as np

eleicoes = pd.read_csv('primary-results.csv')
eleicoes.head()
```

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes
0	Alabama	AL	Autauga	1001.0	Democrat	Bernie Sanders	544	0.182
1	Alabama	AL	Autauga	1001.0	Democrat	Hillary Clinton	2387	0.800
2	Alabama	AL	Baldwin	1003.0	Democrat	Bernie Sanders	2694	0.329
3	Alabama	AL	Baldwin	1003.0	Democrat	Hillary Clinton	5290	0.647
4	Alabama	AL	Barbour	1005.0	Democrat	Bernie Sanders	222	0.078

```
# Analisando os valores únicos de candidatos:
eleicoes['candidate'].unique()
```

```
array(['Bernie Sanders', 'Hillary Clinton', 'Ben Carson', 'Donald Trump',
       'John Kasich', 'Marco Rubio', 'Ted Cruz', ' Uncommitted',
       "Martin O'Malley", 'Carly Fiorina', 'Chris Christie', 'Jeb Bush',
       'Mike Huckabee', 'Rand Paul', 'Rick Santorum', ' No Preference'], dtype=object)
```

Problema

Qual o desempenho de cada candidato, por estado e partido?

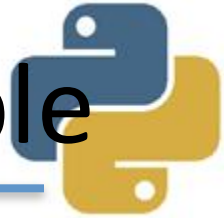
Exercício 98: usando groupby

Qual o desempenho de cada candidato, por estado e partido?

```
# Resposta usando group by
eleicoes.groupby(['state', 'party', 'candidate']).aggregate({'votes':[np.sum]}).head(7)
```

			votes
			sum
state	party	candidate	
Alabama	Democrat	Bernie Sanders	76399
		Hillary Clinton	309928
	Republican	Ben Carson	87517
		Donald Trump	371735
		John Kasich	37970
		Marco Rubio	159802
		Ted Cruz	180608

Exercício 99 : usando pivot_table



Qual o desempenho de cada candidato, por estado e partido?

```
# Resposta usando pivot_table
tab = pd.pivot_table(eleicoes, index=['state', 'party', 'candidate'],
                values=['votes'], aggfunc={'votes':np.sum})
tab.head(7)
```

			votes
state	**party**	**candidate**	
Alabama	**Democrat**	**Bernie Sanders**	76399
		Hillary Clinton	309928
	Republican	**Ben Carson**	87517
		Donald Trump	371735
		John Kasich	37970
		Marco Rubio	159802
		Ted Cruz	180608

Problema

Agora, precisamos saber o ranking das eleições, por partido e distrito

Exercício 100: rank por distrito

```
#Criando um DataFrameGroupBy com Distrito e Partido
grupo = eleicoes.groupby(['county', 'party'])
print 'type(grupo):', type(grupo)
```

```
type(grupo): <class 'pandas.core.groupby.DataFrameGroupBy'>
```

```
# Criando a coluna rank
eleicoes['rank'] = grupo['votes'].rank(ascending=False)
print 'eleicoes["rank"]:', type(eleicoes['rank'])
```

```
eleicoes["rank"]: <class 'pandas.core.series.Series'>
```

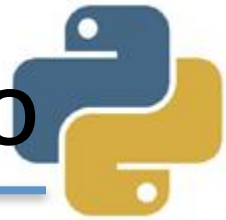
```
#Analisar o resultado da eleição em Los Angeles
eleicoes[eleicoes['county'] == 'Los Angeles']
```

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes	rank
1385	California	CA	Los Angeles	6037.0	Democrat	Bernie Sanders	434656	0.420	2.0
1386	California	CA	Los Angeles	6037.0	Democrat	Hillary Clinton	590502	0.570	1.0
1519	California	CA	Los Angeles	6037.0	Republican	Donald Trump	179130	0.698	1.0
1520	California	CA	Los Angeles	6037.0	Republican	John Kasich	33559	0.131	2.0
1521	California	CA	Los Angeles	6037.0	Republican	Ted Cruz	30775	0.120	3.0

Problema

Agora, precisamos saber o ranking das eleições, por estado, partido e distrito

Exercício 101: rank por distrito



```
# Melhorando o agrupamento: estado, partido e candidato
grupo = eleicoes.groupby(['state', 'party', 'candidate']).sum()
grupo.head()
```

			fips	votes	fraction_votes	rank
state	party	candidate				
Alabama	Democrat	Bernie Sanders	71489.0	76399	12.913	555.0
		Hillary Clinton	71489.0	309928	51.005	266.0
	Republican	Ben Carson	71489.0	87517	6.814	966.0
		Donald Trump	71489.0	371735	32.620	390.0
		John Kasich	71489.0	37970	2.226	1340.0

```
# Melhorando o agrupamento: removendo colunas
del grupo['fips']
del grupo['fraction_votes']
grupo.head()
```

			votes	rank
state	party	candidate		
Alabama	Democrat	Bernie Sanders	76399	555.0
		Hillary Clinton	309928	266.0
	Republican	Ben Carson	87517	966.0
		Donald Trump	371735	390.0
		John Kasich	37970	1340.0

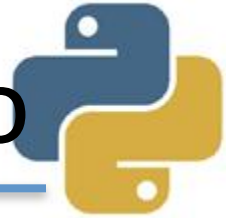
Exercício 102: reset_index

In [966]:
```
# Voltando ao indice padrao
grupo.reset_index(inplace=True)
grupo.head(10)
```

Out[966]:

	state	party	candidate	votes	rank
0	Alabama	Democrat	Bernie Sanders	76399	555.0
1	Alabama	Democrat	Hillary Clinton	309928	266.0
2	Alabama	Republican	Ben Carson	87517	966.0
3	Alabama	Republican	Donald Trump	371735	390.0
4	Alabama	Republican	John Kasich	37970	1340.0
5	Alabama	Republican	Marco Rubio	159802	805.5
6	Alabama	Republican	Ted Cruz	180608	680.0
7	Alaska	Democrat	Bernie Sanders	440	40.0
8	Alaska	Democrat	Hillary Clinton	99	80.0
9	Alaska	Republican	Ben Carson	2401	150.5

Exercício 103: rank por estado



```
# Criando o rank por estado e partido
grupo['rank'] = grupo.groupby(['state', 'party'])['votes'].rank(ascending=False)
grupo.head(7)
```

	state	party	candidate	fips	votes	fraction_votes	rank
0	Alabama	Democrat	Bernie Sanders	71489.0	76399	12.913	2.0
1	Alabama	Democrat	Hillary Clinton	71489.0	309928	51.005	1.0
2	Alabama	Republican	Ben Carson	71489.0	87517	6.814	4.0
3	Alabama	Republican	Donald Trump	71489.0	371735	32.620	1.0
4	Alabama	Republican	John Kasich	71489.0	37970	2.226	5.0
5	Alabama	Republican	Marco Rubio	71489.0	159802	9.996	3.0
6	Alabama	Republican	Ted Cruz	71489.0	180608	13.773	2.0

Exercício 104: pivot_table

```
# Vamos sumarizar os dados por estado, partido e candidato
pd.pivot_table(grupo, index=['state', 'party', 'candidate'],
                      values=['rank', 'votes']).head()
```

			rank	votes
state	**party**	**candidate**		
Alabama	**Democrat**	**Bernie Sanders**	2.0	76399
		Hillary Clinton	1.0	309928
	Republican	**Ben Carson**	4.0	87517
		Donald Trump	1.0	371735
		John Kasich	5.0	37970

Problema

Em QUAIS estados cada candidato ganhou?

Exercício 105: rank==1

```
# Em QUAIS estados cada candidato ganhou?
grupo[grupo['rank'] == 1].head()
```

	state	party	candidate	votes	rank
1	Alabama	Democrat	Hillary Clinton	309928	1.0
3	Alabama	Republican	Donald Trump	371735	1.0
7	Alaska	Democrat	Bernie Sanders	440	1.0
13	Alaska	Republican	Ted Cruz	7973	1.0
15	Arizona	Democrat	Hillary Clinton	235697	1.0

Problema

Em QUANTOS estados cada candidato ganhou?

Exercício 106: value_counts()

```
# Em QUANTOS estados cada candidato ganhou?
grupo[grupo['rank'] == 1]['candidate'].value_counts()
```

```
Donald Trump         36
Hillary Clinton      28
Bernie Sanders       21
Ted Cruz              9
John Kasich           1
Name: candidate, dtype: int64
```

Visualização Matplotlib

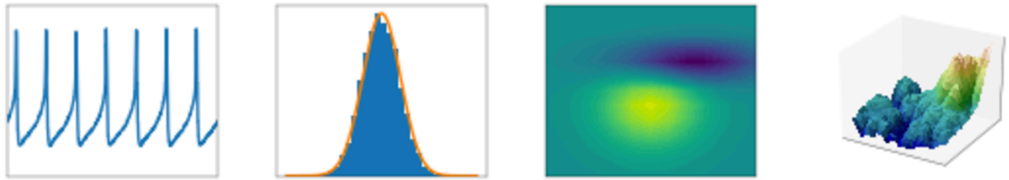
matplotlib.org

matplotlib

home | examples | gallery | pyplot | docs »

Introduction

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.



Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For a sampling, see the screenshots, thumbnail gallery, and examples directory

For simple plotting the `pyplot` module provides a MATLAB-like interface, particularly when combined with `IPython`. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Exercício 107: np.linspace

Mais um pouco sobre visualização com matplotlib

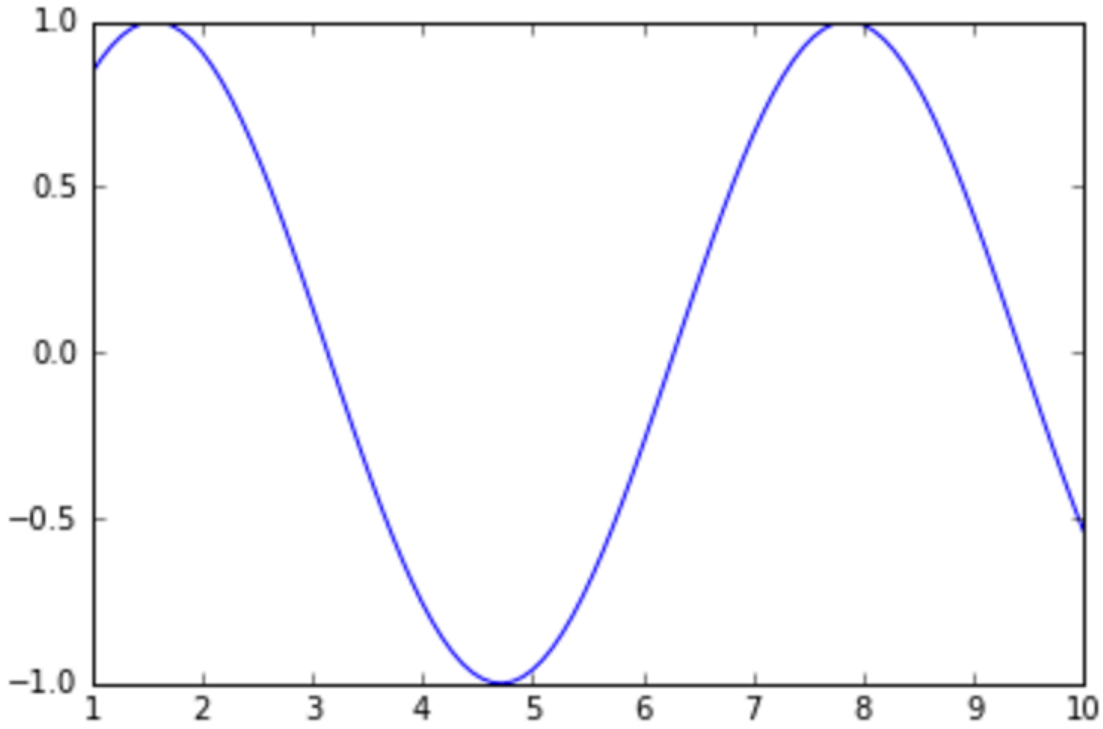
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# Vamos gerar um array com 100 valores, entre 1 e 10
x = np.linspace(1, 10, 100)
x
```

```
array([  1.        ,   1.09090909,   1.18181818,   1.27272727,
         1.36363636,   1.45454545,   1.54545455,   1.63636364,
         1.72727273,   1.81818182,   1.90909091,   2.        ,
         2.09090909,   2.18181818,   2.27272727,   2.36363636,
         2.45454545,   2.54545455,   2.63636364,   2.72727273,
         2.81818182,   2.90909091,   3.        ,   3.09090909,
         3.18181818,   3.27272727,   3.36363636,   3.45454545,
         3.54545455,   3.63636364,   3.72727273,   3.81818182,
         3.90909091,   4.        ,   4.09090909,   4.18181818,
         4.27272727,   4.36363636,   4.45454545,   4.54545455,
         4.63636364,   4.72727273,   4.81818182,   4.90909091,
         5.        ,   5.09090909,   5.18181818,   5.27272727,
         5.36363636,   5.45454545,   5.54545455,   5.63636364,
         5.72727273,   5.81818182,   5.90909091,   6.        ,
```

Exercício 108: np.sin(x)

```
#Plotando os eixos X e Y(sin(x))
plt.plot(x, np.sin(x))
```

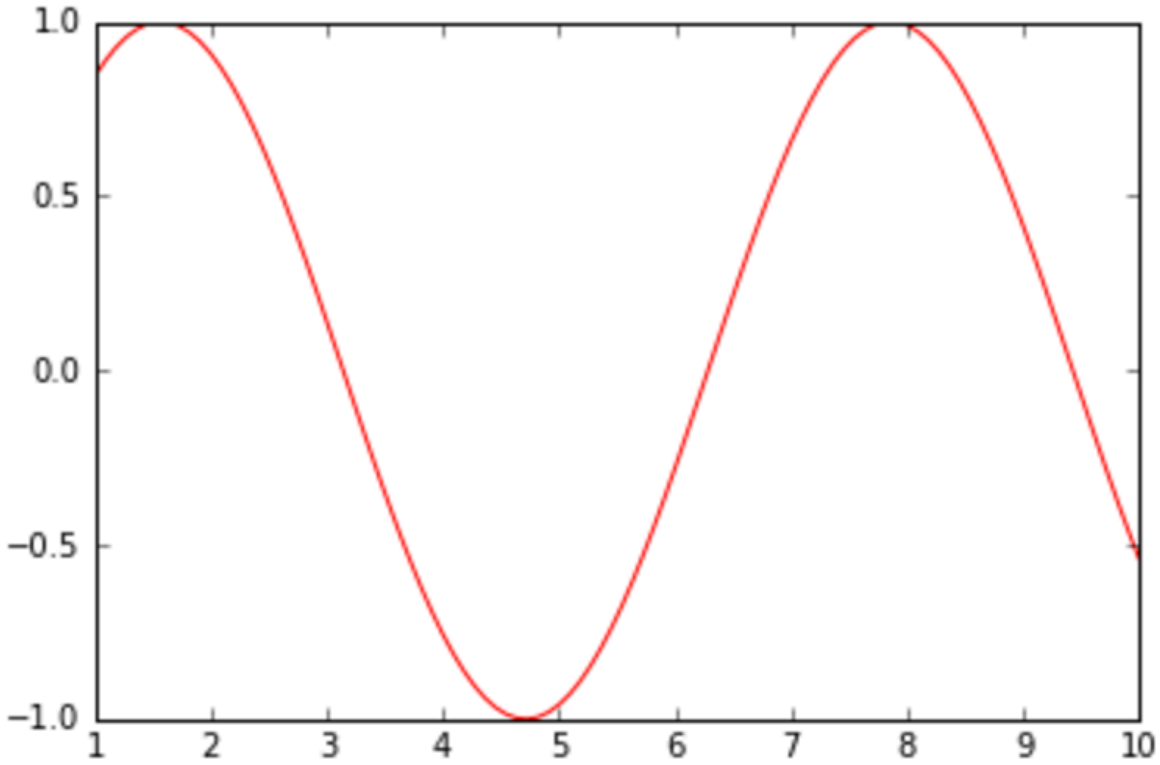
```
[<matplotlib.lines.Line2D at 0x127a77050>]
```



Exercício 109: atalho para cor ‘r’

```
plt.plot(x, np.sin(x), 'r')
```

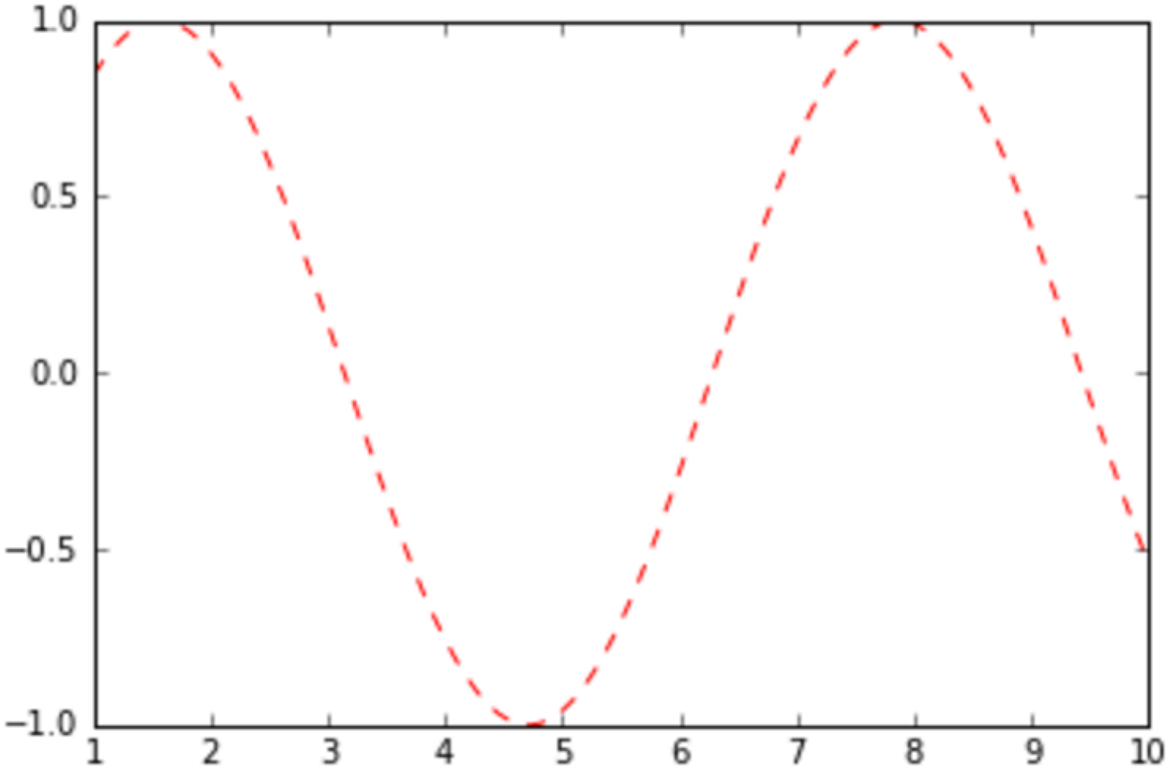
```
[<matplotlib.lines.Line2D at 0x12be8b210>]
```



Exercício 110: atalho cor,estilo ‘r--’

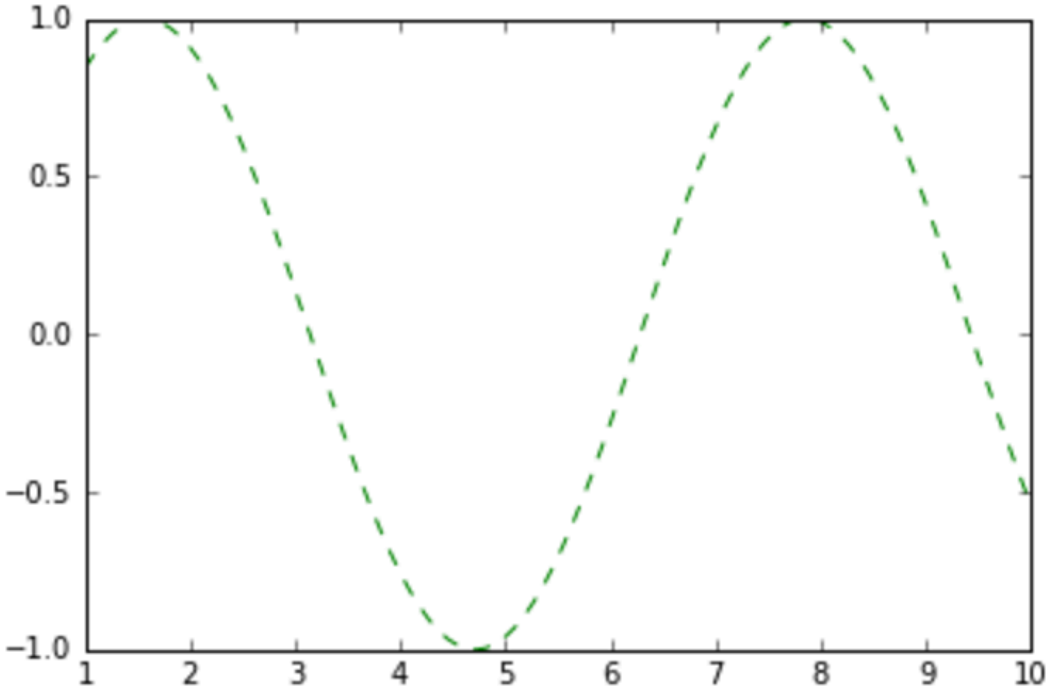
```
plt.plot(x, np.sin(x), 'r--')
```

```
[<matplotlib.lines.Line2D at 0x12d1a6950>]
```

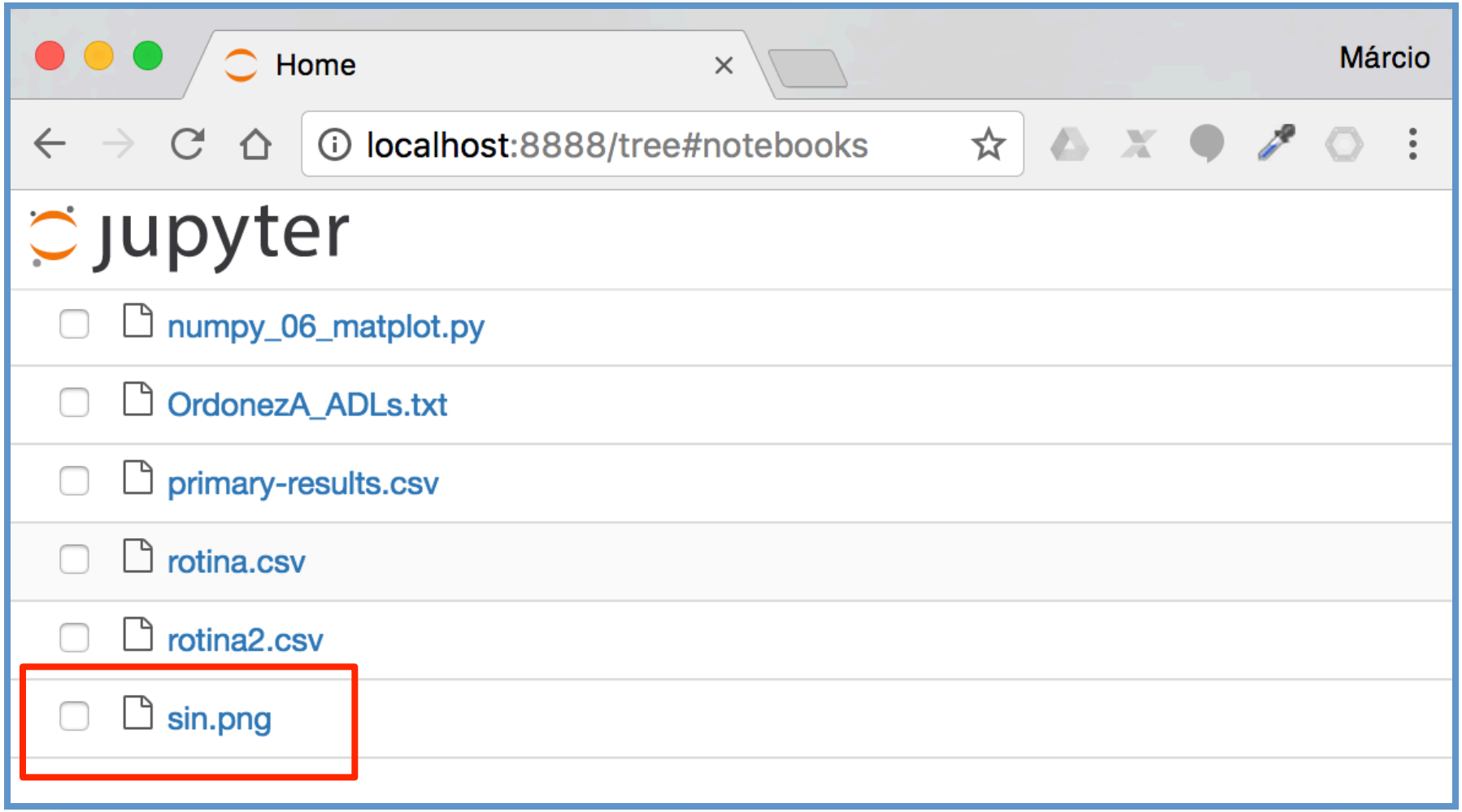


Exercício 111: salvar figura–savefig

```
# Referência para os dados da figura gerada
fig = plt.figure()
plt.plot(x, np.sin(x), 'g--')
# Salvar a figura 'sin.png' no diretório do notebook
fig.savefig('sin.png')
```



Arquivo gerado

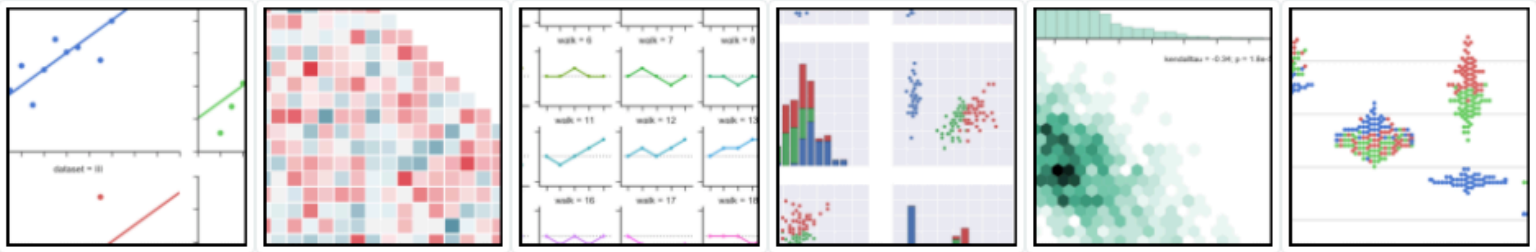


Visualização Seaborn

seaborn.pydata.org

seaborn 0.7.1 API Tutorial Gallery Site Page Search

Seaborn: statistical data visualization



Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.

For a brief introduction to the ideas behind the package, you can read the introductory notes. More practical information is on the installation page. You may also want to browse the example gallery to get a sense for what you can do with seaborn and then check out the tutorial and API reference to find out how.

To see the code or report a bug, please visit the github repository. General support issues are most at home on stackoverflow, where there is a seaborn tag.

Documentation

- An introduction to seaborn
- What's new in the package
- Installing and getting started
- Example gallery
- API reference
- Seaborn tutorial

Features

- Style functions: API | Tutorial
- Color palettes: API | Tutorial
- Distribution plots: API | Tutorial
- Regression plots: API | Tutorial
- Categorical plots: API | Tutorial
- Axis grid objects: API | Tutorial

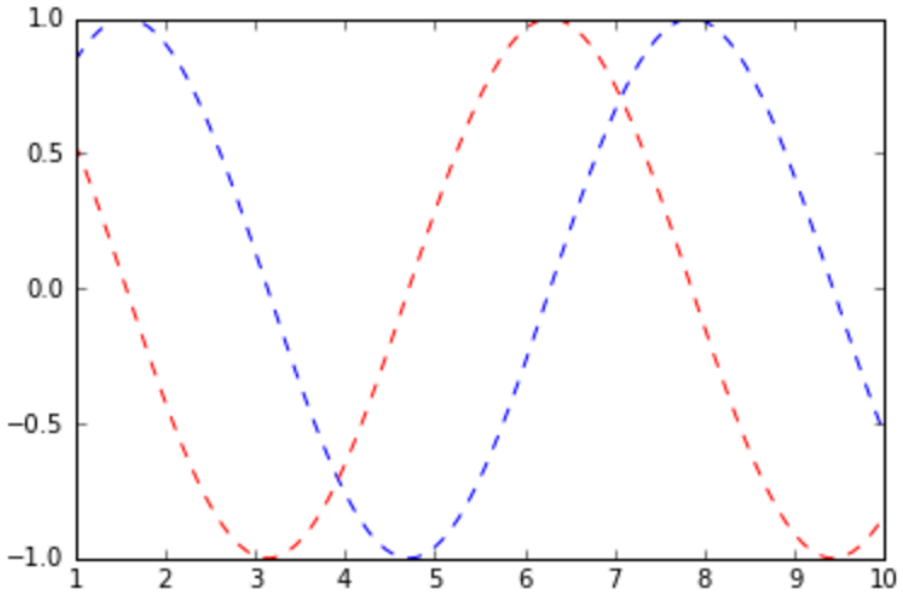
Visualização Seaborn

- O matplotlib é a ferramenta padrão para visualização, por implementar as funções essenciais à plotagem de gráficos
- O Seaborn é usa o matplotlib e melhora aspectos visuais dos gráficos gerados
- $ pip install seaborn

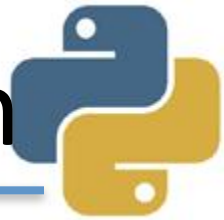
Exercício 112: plot padrão

Visualização com Seaborn

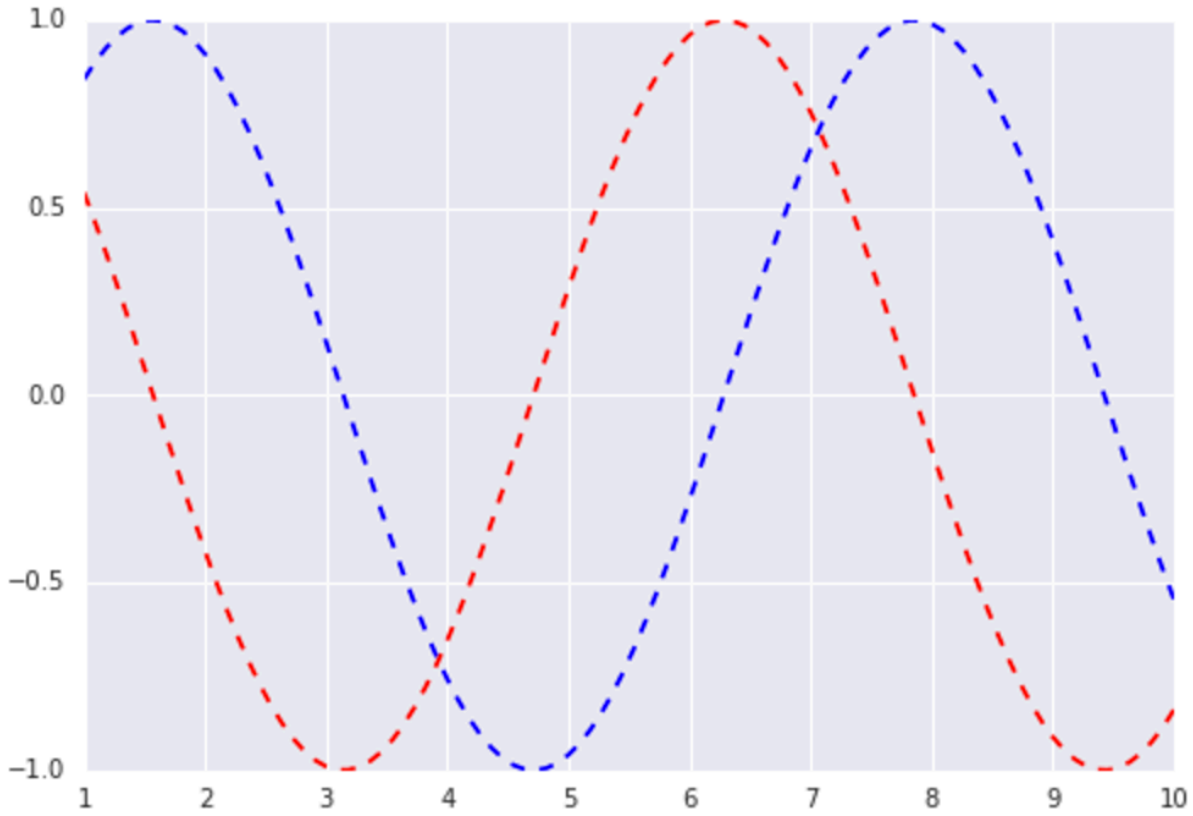
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(1, 10, 100)
plt.plot(x, np.sin(x), 'b--')
plt.plot(x, np.cos(x), 'r--')
plt.show()
```



Exercício 113: import seaborn

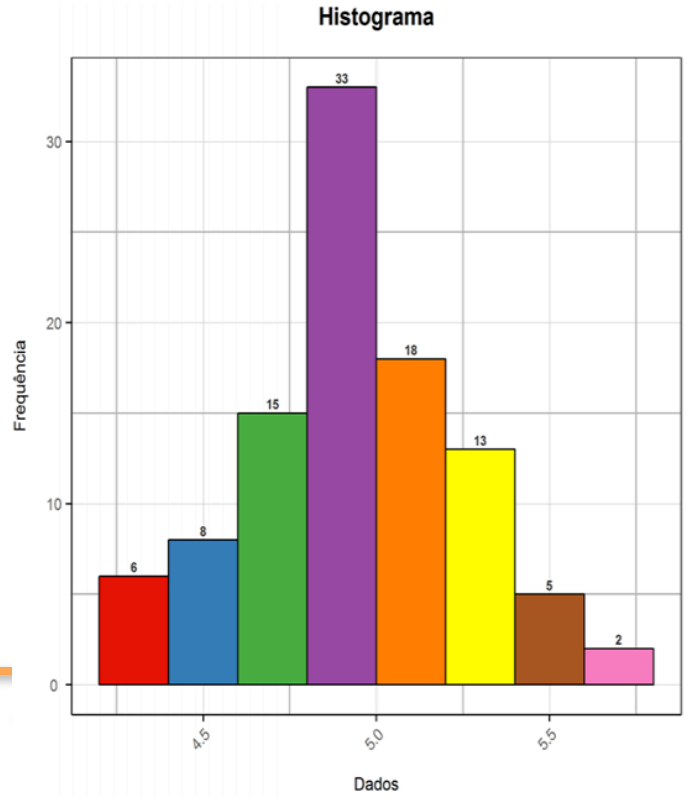


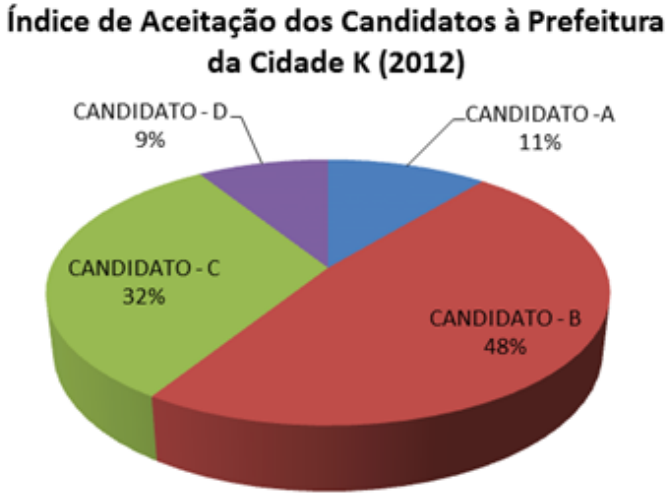
```
import seaborn as sns

plt.plot(x, np.sin(x), 'b--')
plt.plot(x, np.cos(x), 'r--')
plt.show()
```



Histograma e gráfico de pizza

- Gráficos muito utilizados para estudos de distribuição de frequências





Exercício 114: titanic docs

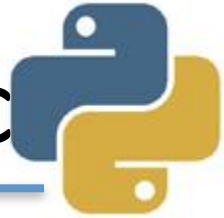
Histograma e gráfico de pizza

```
%matplotlib inline
import pandas as pd
import pydataset
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Visualizando a documentação do dataset Titanic
pydataset.data('titanic', show_doc=True)
```

```
titanic

PyDataset Documentation (adopted from R Documentation. The displayed examples are in R)

## titanic

### Description

The data is an observation-based version of the 1912 Titanic passenger
survival log,

### Usage
```

Exercício 115: carregar titanic



In [1051]:
```
titanic = pydataset.data('titanic')
titanic.head()
```

Out[1051]:

	class	age	sex	survived
1	1st class	adults	man	yes
2	1st class	adults	man	yes
3	1st class	adults	man	yes
4	1st class	adults	man	yes
5	1st class	adults	man	yes

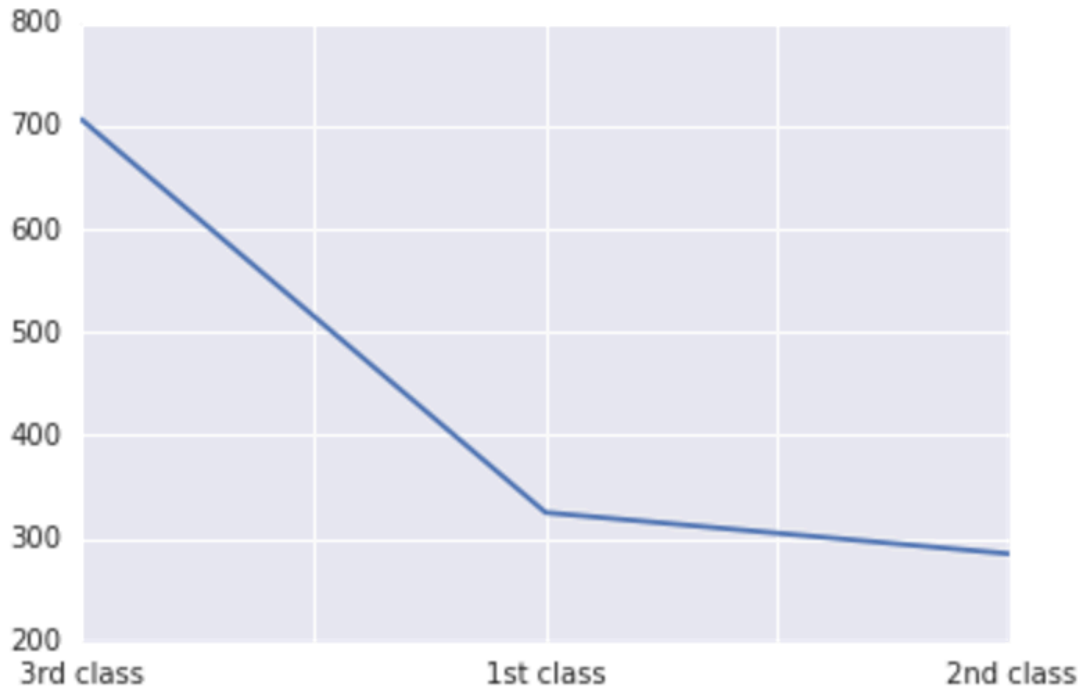
Exercício 116: pessoas por classe

```
# Visualização números de pessoas por class
titanic['class'].value_counts()
```

```
3rd class    706
1st class    325
2nd class    285
Name: class, dtype: int64
```

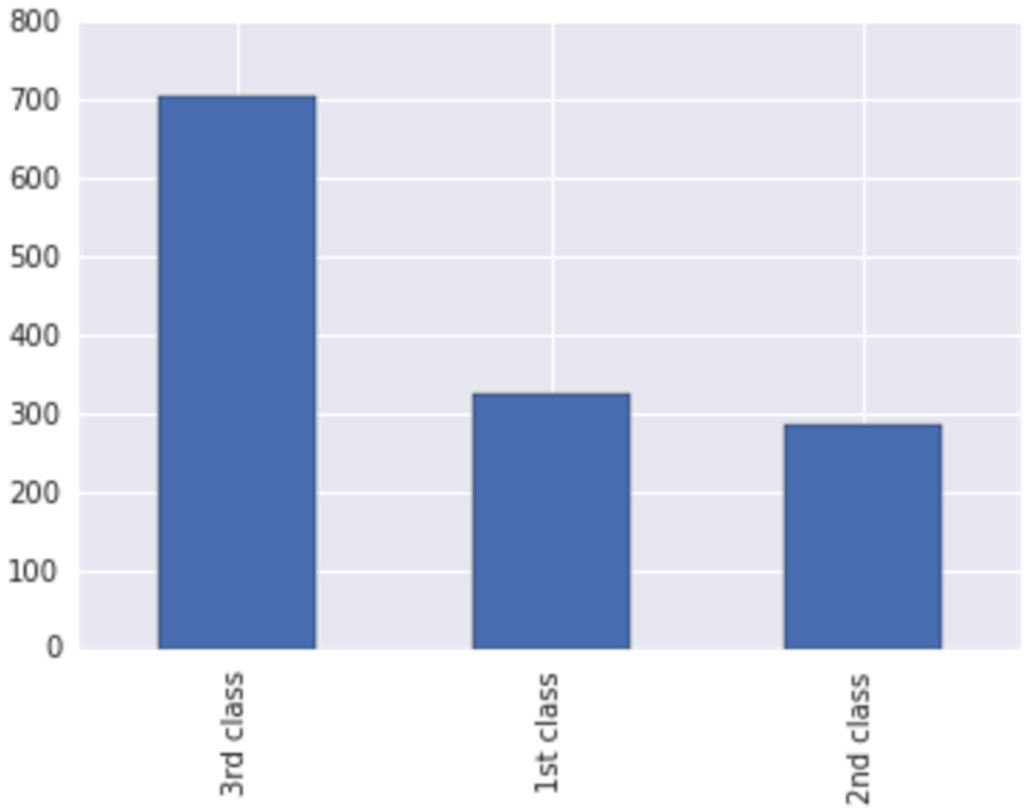
Exercício 117: gráfico freq. classe

```
# Gráfico mostrando a frequência por classe
titanic['class'].value_counts().plot()
plt.show()
```



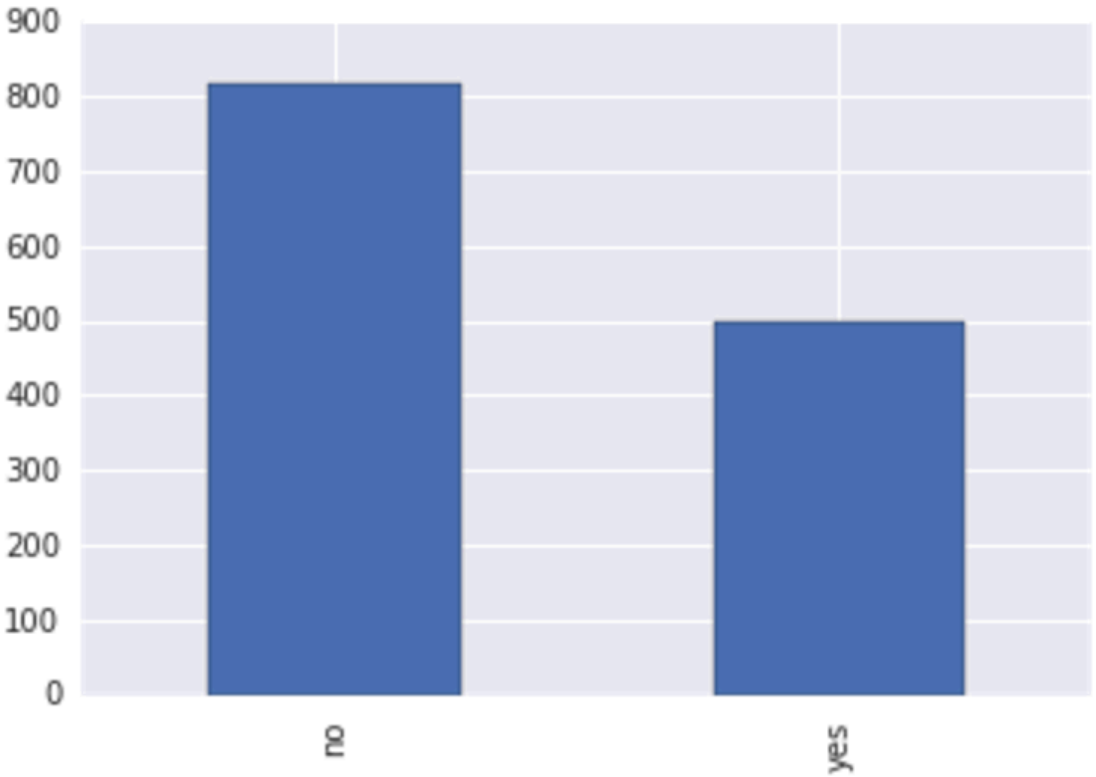
Exercício 118: histograma

```
# Histograma mostrando a frequência por classe
titanic['class'].value_counts().plot(kind='bar')
plt.show()
```



Exercício 119: Sobreviventes

```
# Plot da coluna 'survived'
titanic['survived'].value_counts().plot(kind='bar')
plt.show()
```



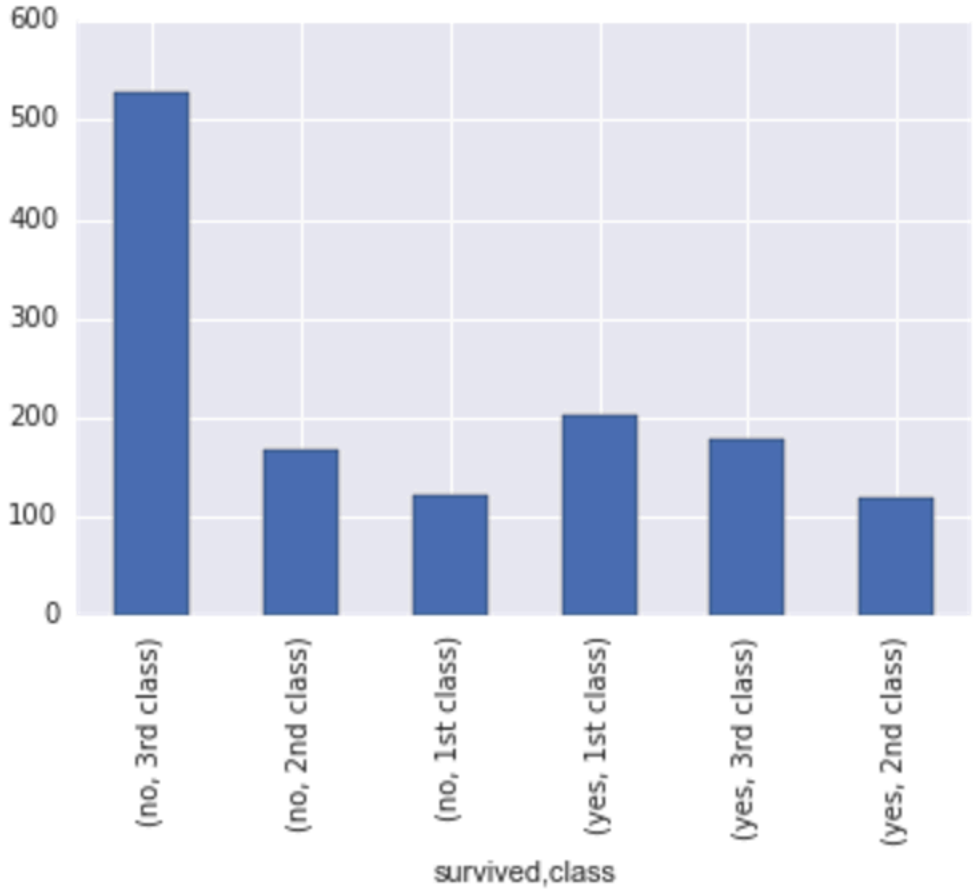
Exercício 120: combinando

```
#Verificando os sobreviventes por classe
titanic.groupby('survived')['class'].value_counts()
```

```
survived  class
no        3rd class    528
          2nd class    167
          1st class    122
yes       1st class    203
          3rd class    178
          2nd class    118
dtype: int64
```

Exercício 121: plot combinando

```
#Plot da combinação: 'class' e 'survived'
titanic.groupby('survived')['class'].value_counts().plot(kind='bar')
plt.show()
```



Gráficos de pizza

- Gráficos muito usados para representação distribuição de frequência dos dados
- Trabalham com valores percentuais
- Vamos carregar o dataset com dados da população do Brasil, por estado

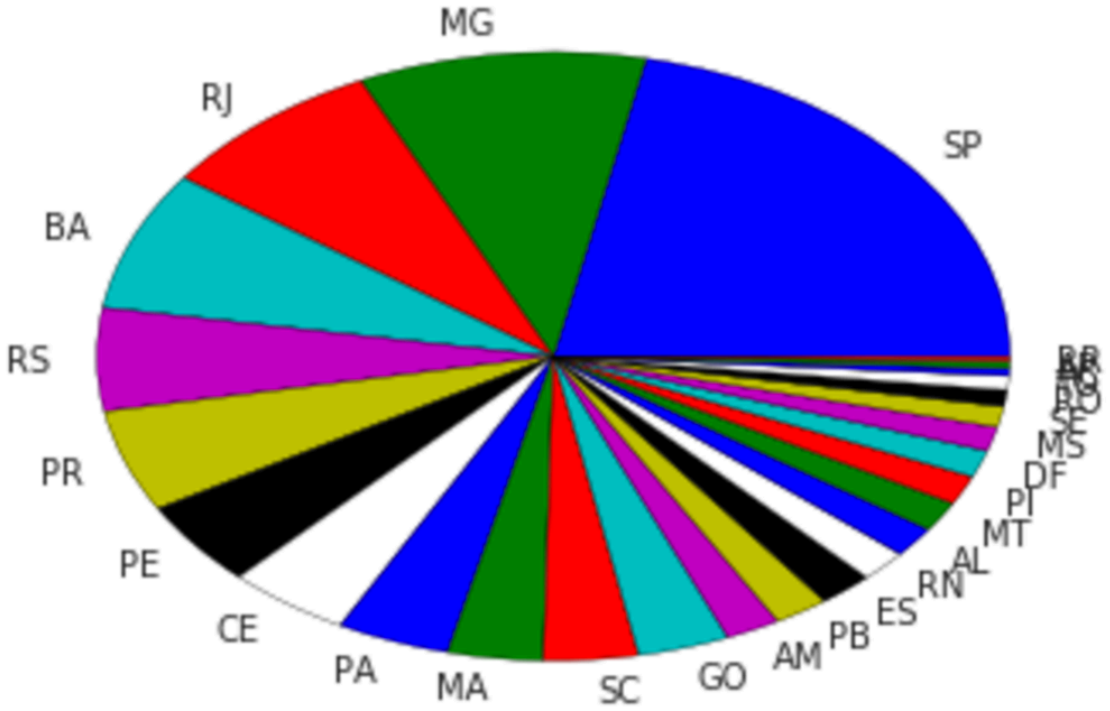
Exercício 122: população brasileira

```
# Carga dos dados da população brasileira
estados = pd.read_csv('populacao_brasil.csv')
estados.head()
```

	estado	total
0	SP	44846530
1	MG	21024678
2	RJ	16690709
3	BA	15276566
4	RS	11286500

Exercício 123: plot população

```
# Gráfico de pizza
plt.pie(estados['percent'], labels=estados['estado'])
plt.show()
```



Exercício 124: autopct

```
# Gráfico de pizza - mostrando os percentuais
plt.pie(estados['percent'], labels=estados['estado'], autopct='%1.2f%%')
plt.show()
```

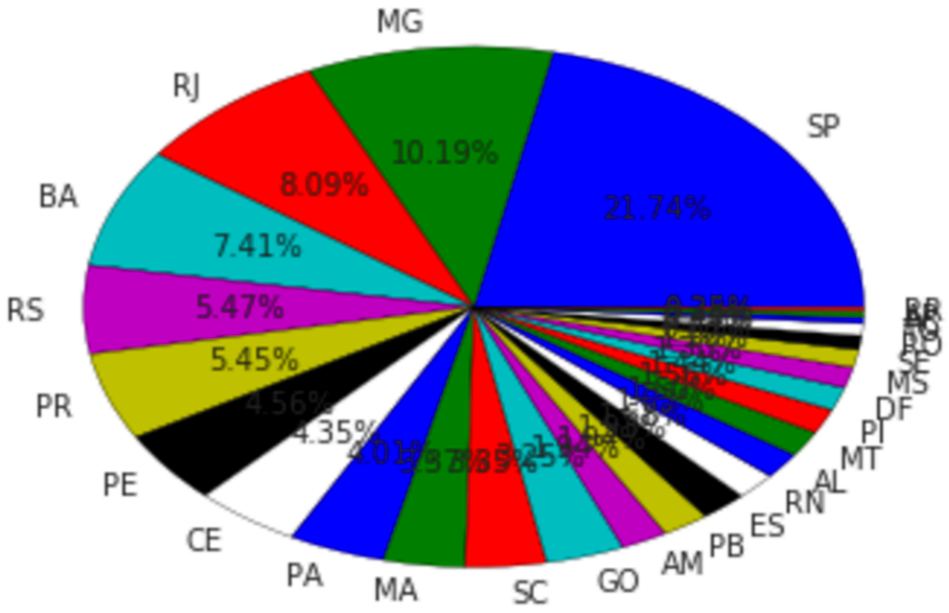
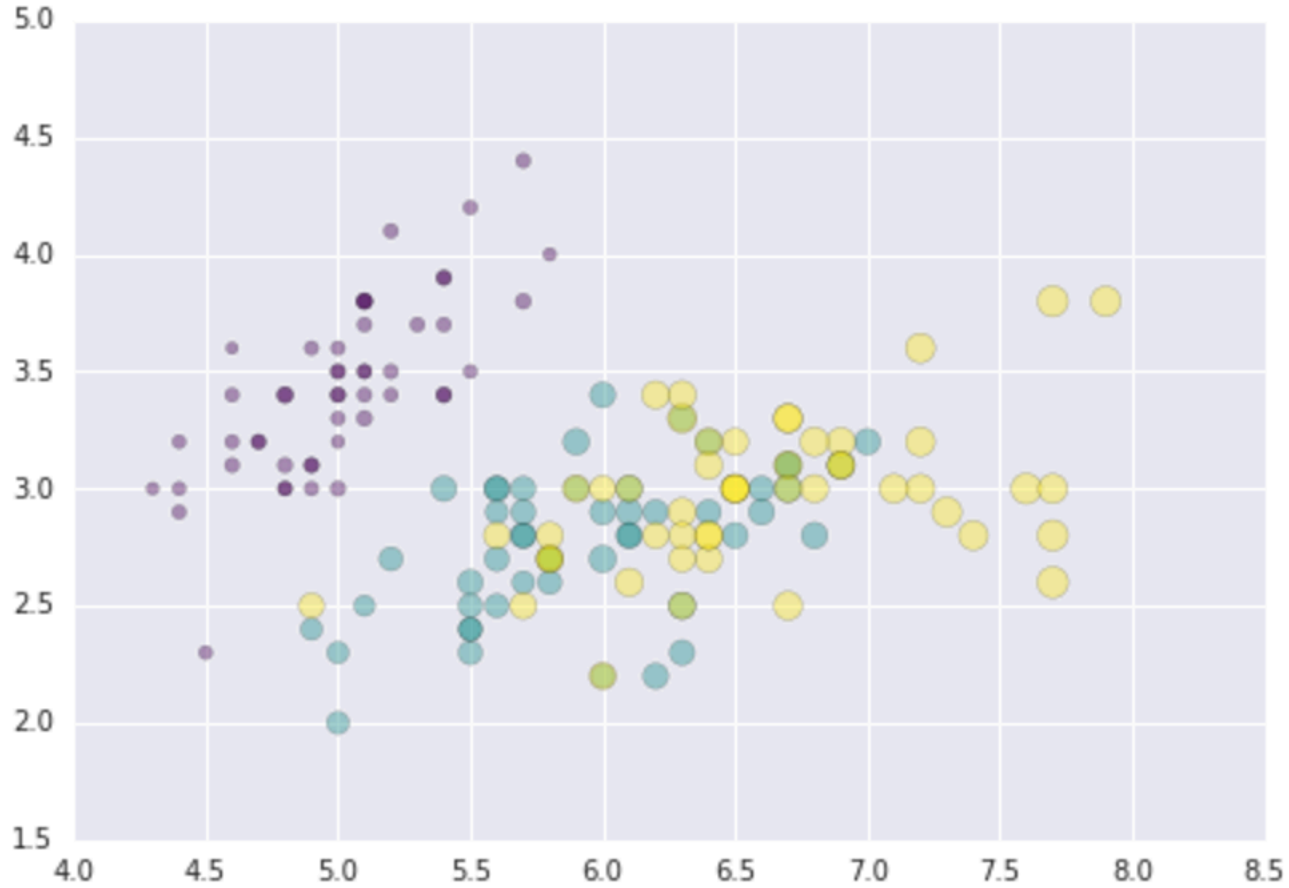


Gráfico de dispersão



Gráficos de dispersão

- São representações de duas ou mais variáveis que são organizadas em um gráfico, uma em função da outra.
- Foi descrito pela primeira vez por Francis Galton
- Vamos usar o a base do pydataset: AirPassengers

Exercício 125: AirPassengers

Gráficos de dispersão

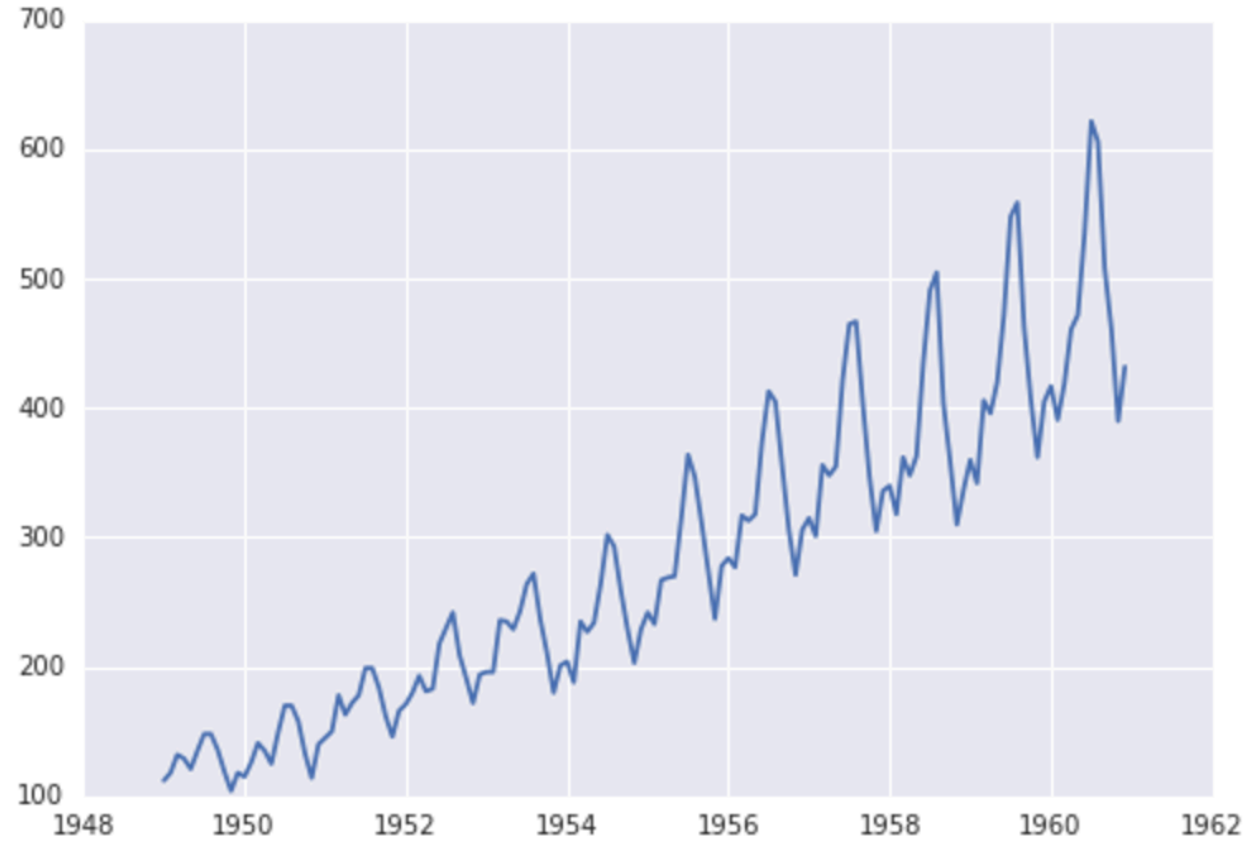
In [3]:
```
%matplotlib inline
import pandas as pd
import pydataset
import matplotlib.pyplot as plt
import seaborn as sns

# Carga do dataset
passageiros = pydataset.data('AirPassengers')
passageiros.head()
```

Out[3]:

	time	AirPassengers
1	1949.000000	112
2	1949.083333	118
3	1949.166667	132
4	1949.250000	129
5	1949.333333	121

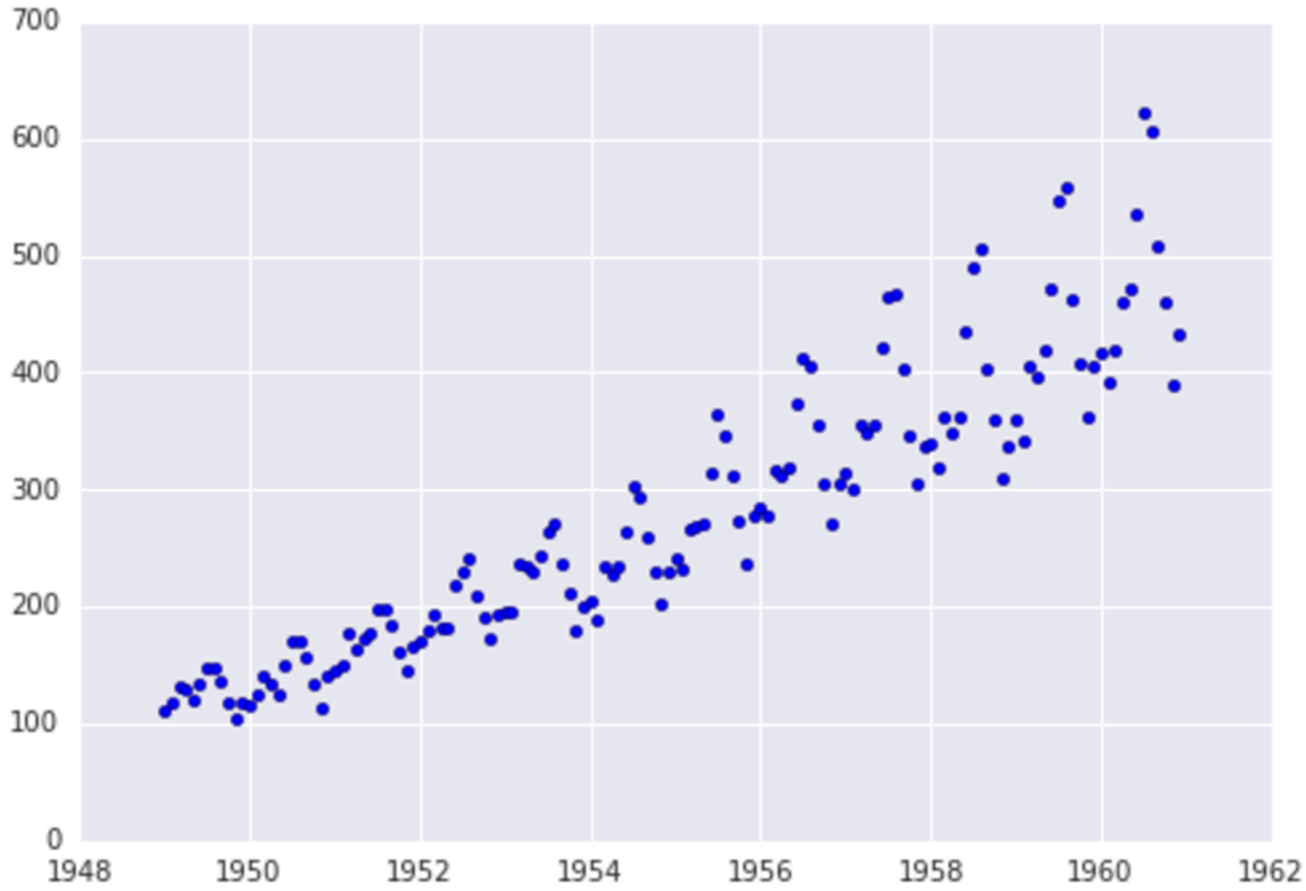
Exercício 126: plt.plot

```
# Gráfico usando plot: Passageiros ao longo do meses
# plt.plot - tenta ligar os pontos
plt.plot(passageiros['time'], passageiros['AirPassengers'])
plt.show()
```



Exercício 127: plt.scatter

```
# Gráfico usando scatter:  Passageiros ao longo do meses
plt.scatter(passageiros['time'], passageiros['AirPassengers'])
plt.show()
```



Exercício 128: Iris dataset

In [7]:
```
# Usando o famoso dataset 'iris'
iris = pydataset.data('iris')
iris.head()
```

Out[7]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa

In [8]:
```
# Mostrando o final do dataset
iris.tail()
```

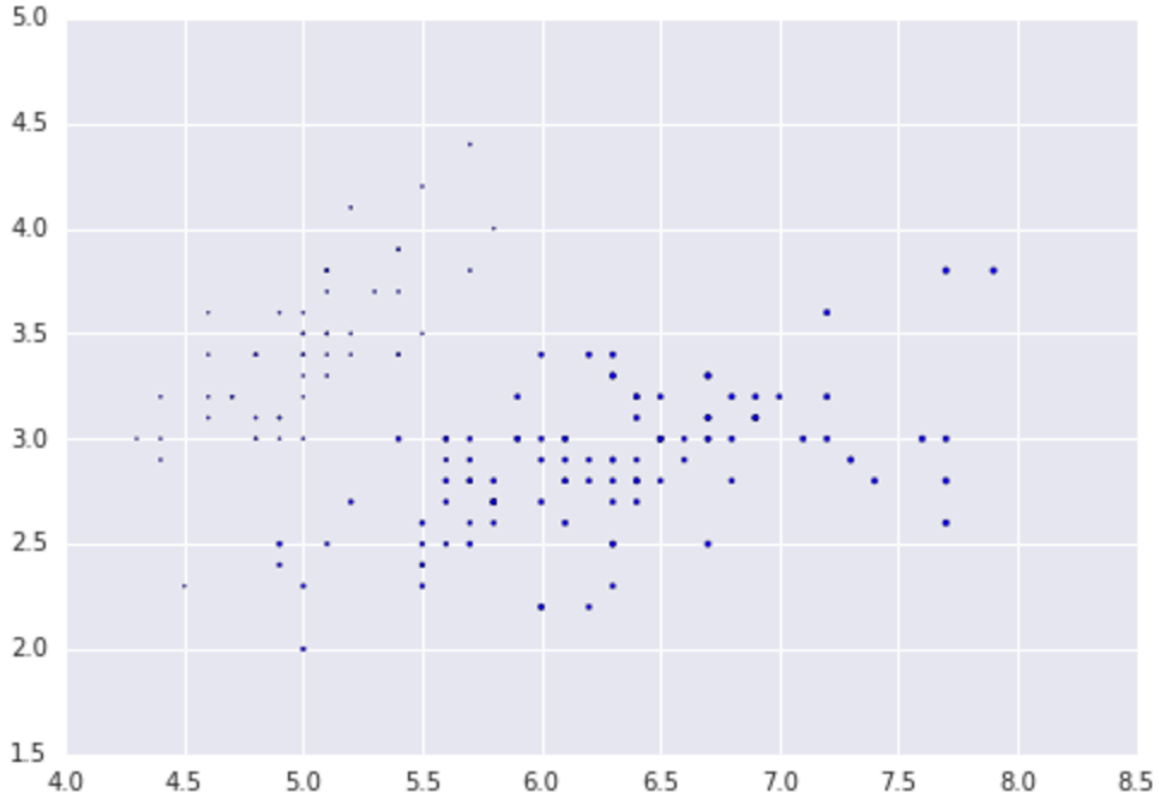
Out[8]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

Exercício 129: sepal e petal

```
# Plotando as colunas Sepal.Length e Sepal.Width, em função de Petal.Length
plt.scatter(iris['Sepal.Length'], iris['Sepal.Width'], sizes=iris['Petal.Length'])
```

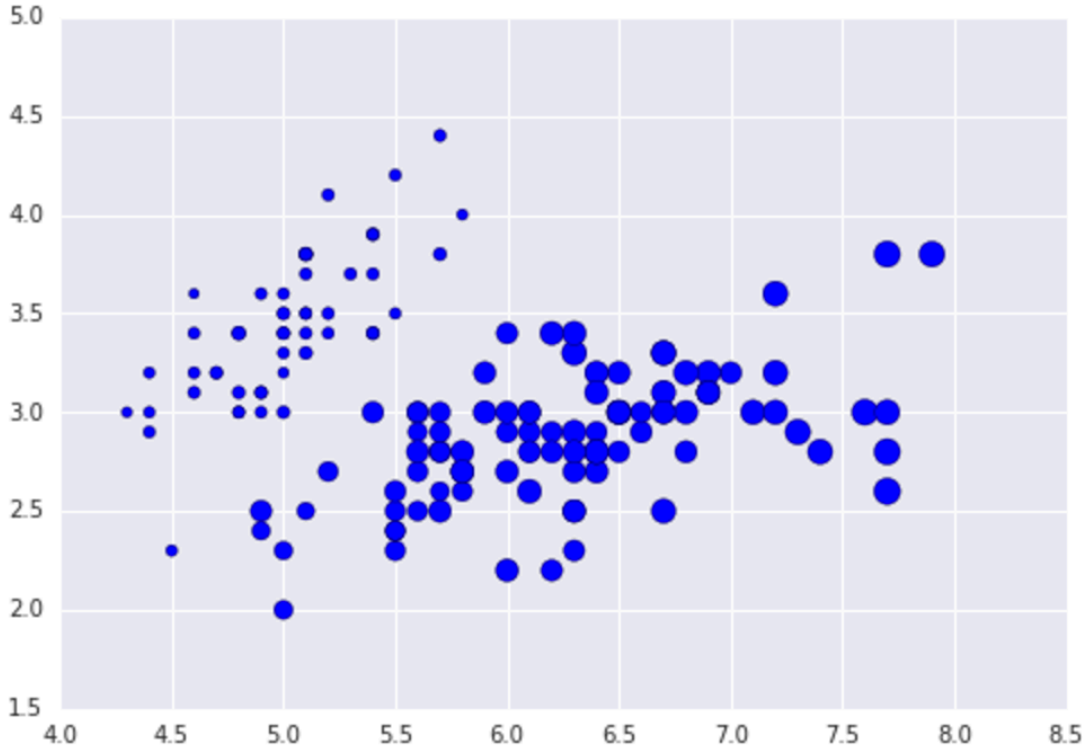
```
<matplotlib.collections.PathCollection at 0x1135fef90>
```



Exercício 130: sizes

```
# Aumentando o tamanho dos pontos: sizes=20*iris['Petal.Length']
plt.scatter(iris['Sepal.Length'], iris['Sepal.Width'], sizes=20*iris['Petal.Length'])
```

```
<matplotlib.collections.PathCollection at 0x11413d050>
```



Exercício 131: nova coluna

```
# Definição da função para escoha da cor
def specie_color(x):
    if x == 'setosa':
        return 0
    elif x== 'versicolor':
        return 1
    return 2
```

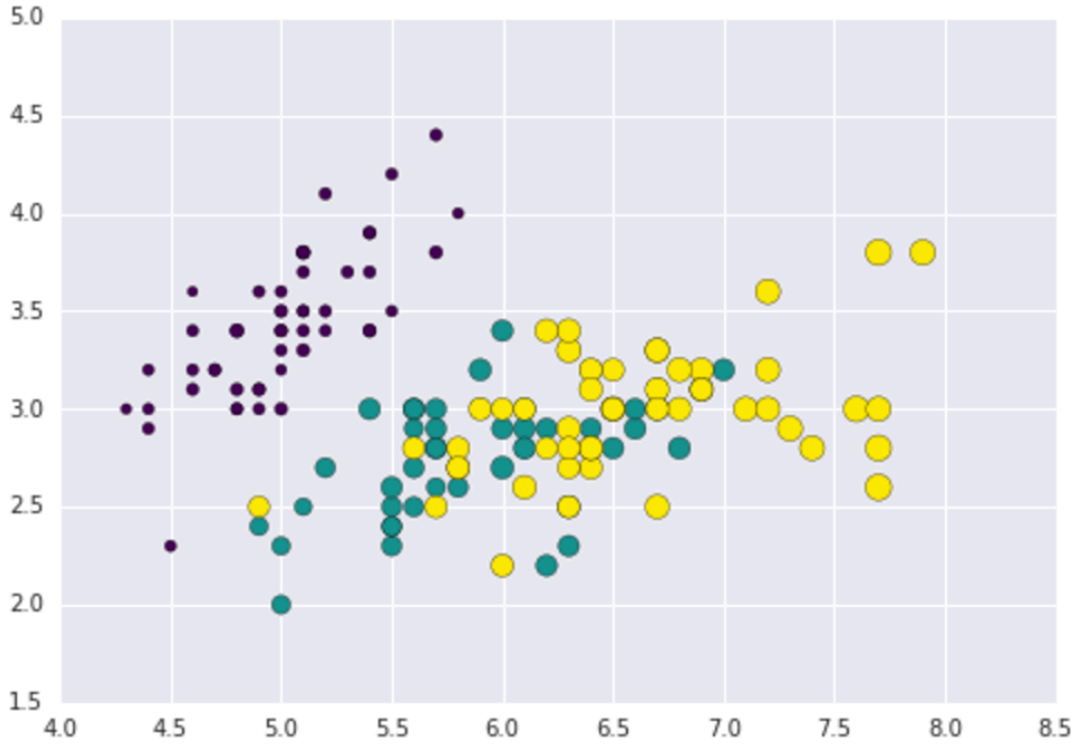
```
# Criação da coluna SpeciesNumber
iris['SpeciesNumber'] = iris['Species'].apply(specie_color)
iris.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	SpeciesNumber
1	5.1	3.5	1.4	0.2	setosa	0
2	4.9	3.0	1.4	0.2	setosa	0
3	4.7	3.2	1.3	0.2	setosa	0
4	4.6	3.1	1.5	0.2	setosa	0
5	5.0	3.6	1.4	0.2	setosa	0

Exercício 132: cores e cmap

```
# Definição de cores: c=iris['SpeciesNumber'], cmap='viridis'
plt.scatter(
    iris['Sepal.Length'], iris['Sepal.Width'], sizes=20*iris['Petal.Length'],
    c=iris['SpeciesNumber'], cmap='viridis'
)
```

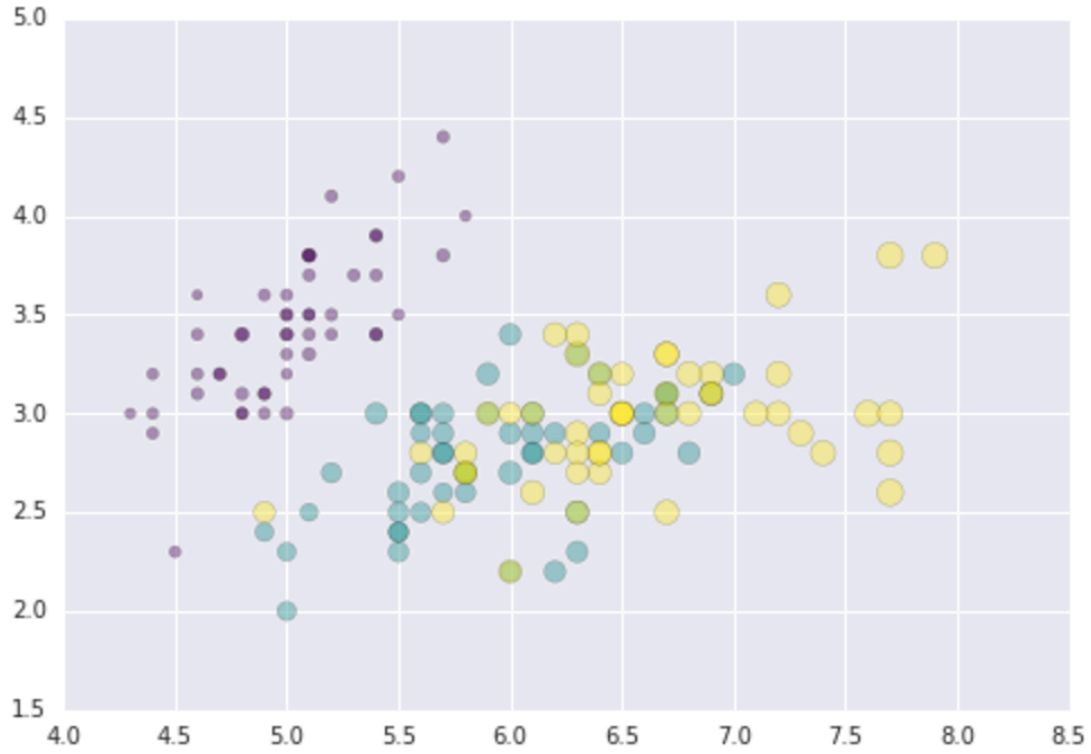
```
<matplotlib.collections.PathCollection at 0x113457f10>
```



Exercício 133: transparência

```
# Informando a transparência: alpha=0.4
plt.scatter(
    iris['Sepal.Length'], iris['Sepal.Width'], sizes=20*iris['Petal.Length'],
    c=iris['SpeciesNumber'], cmap='viridis', alpha=0.4
)
```

```
<matplotlib.collections.PathCollection at 0x11500bd50>
```

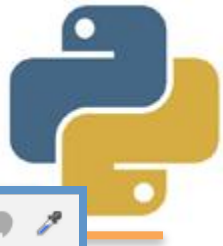


Visualização em mapas

- Vamos continuar trabalhando com gráficos de dispersão
- Mas agora queremos mostrar a dispersão de coordenadas de latitude e longitude em um mapa
- Vamos usar copacabana_lat_lng.csv

Framework JavaScript

Biblioteca Python para mapas



GitHub, Inc. [US] https://github.com/jwass/mplleaflet

mplleaflet

$ pip install mplleaflet

mplleaflet is a Python library that converts a matplotlib plot into a webpage containing a pannable, zoomable Leaflet map. It can also embed the Leaflet map in an IPython notebook. The goal of mplleaflet is to enable use of Python and matplotlib for visualizing geographic data on slippy maps without having to write any Javascript or HTML. You also don't need to worry about choosing the base map content i.e., coastlines, roads, etc.

Only one line of code is needed to convert a plot into a web map. `mplleaflet.show()`

The library is heavily inspired by mpld3 and uses mplexporter to do most of the heavy lifting to walk through Figure objects.

Examples

Basic usage

The simplest use is to just create your plot using matplotlib commands and call `mplleaflet.show()`.

```
>>> import matplotlib.pyplot as plt
... # Load longitude, latitude data
>>> plt.hold(True)
# Plot the data as a blue line with red squares on top
# Just plot longitude vs. latitude
>>> plt.plot(longitude, latitude, 'b') # Draw blue line
>>> plt.plot(longitude, latitude, 'rs') # Draw red squares
```

Exercício 134: copacabana

Visualização em mapas

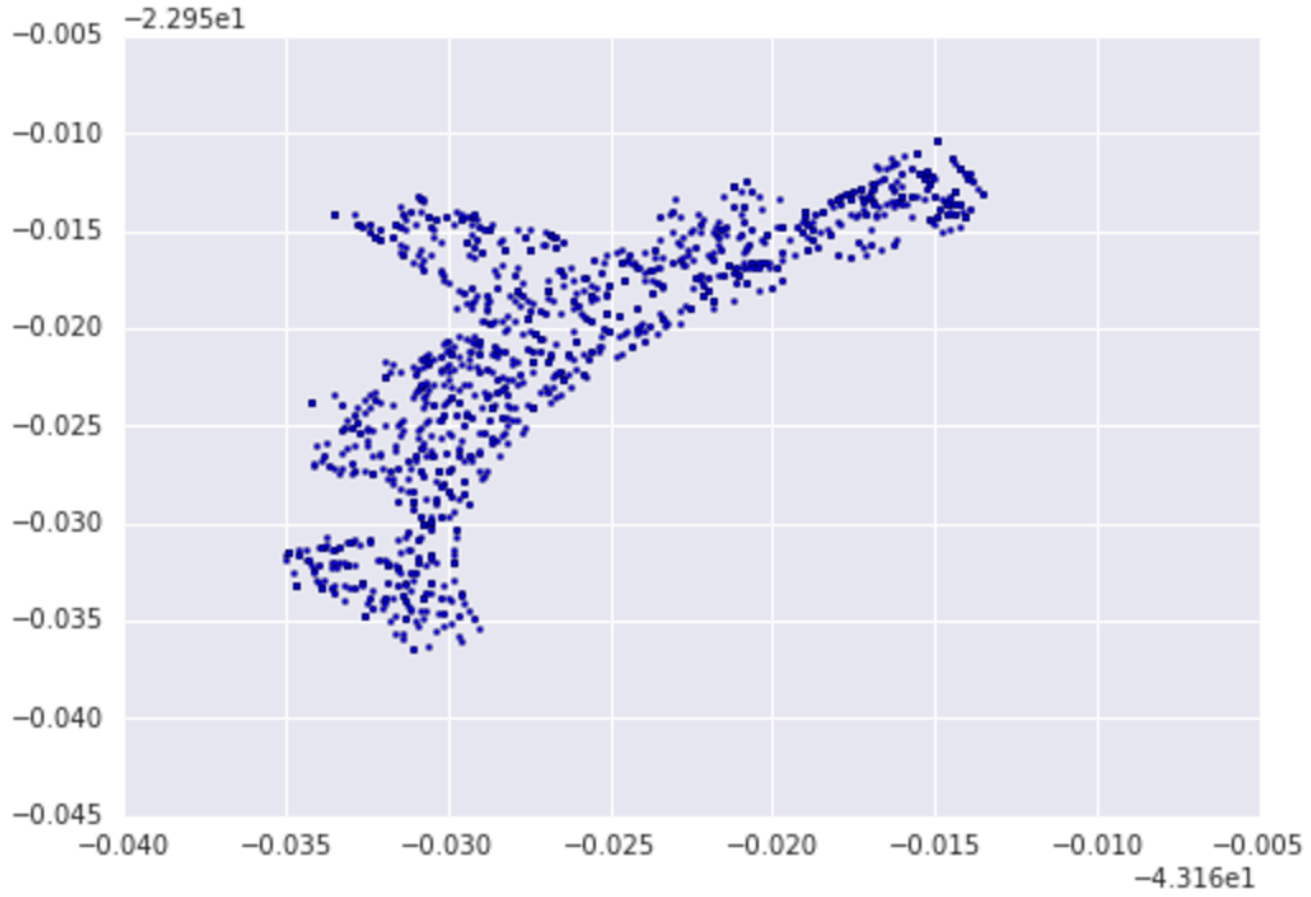
```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Carga do dataset
df = pd.read_csv('copacabana_lat_lng.csv')
df.head()
```

de	Tipologia	AreaConstruida	VAL_UNIT	X	Y	lat	lng
)	1	95	4379	685365.0700	7457802.680	-22.977728	-43.191741
)	1	71	6479	685941.5500	7459001.320	-22.966842	-43.186264
)	1	58	12414	685627.3900	7459080.520	-22.966162	-43.189337
)	1	88	11250	685438.2001	7458268.280	-22.973516	-43.191084
)	1	68	13382	685764.3840	7458954.513	-22.967284	-43.187986

Exercício 135: scatter lng, lat

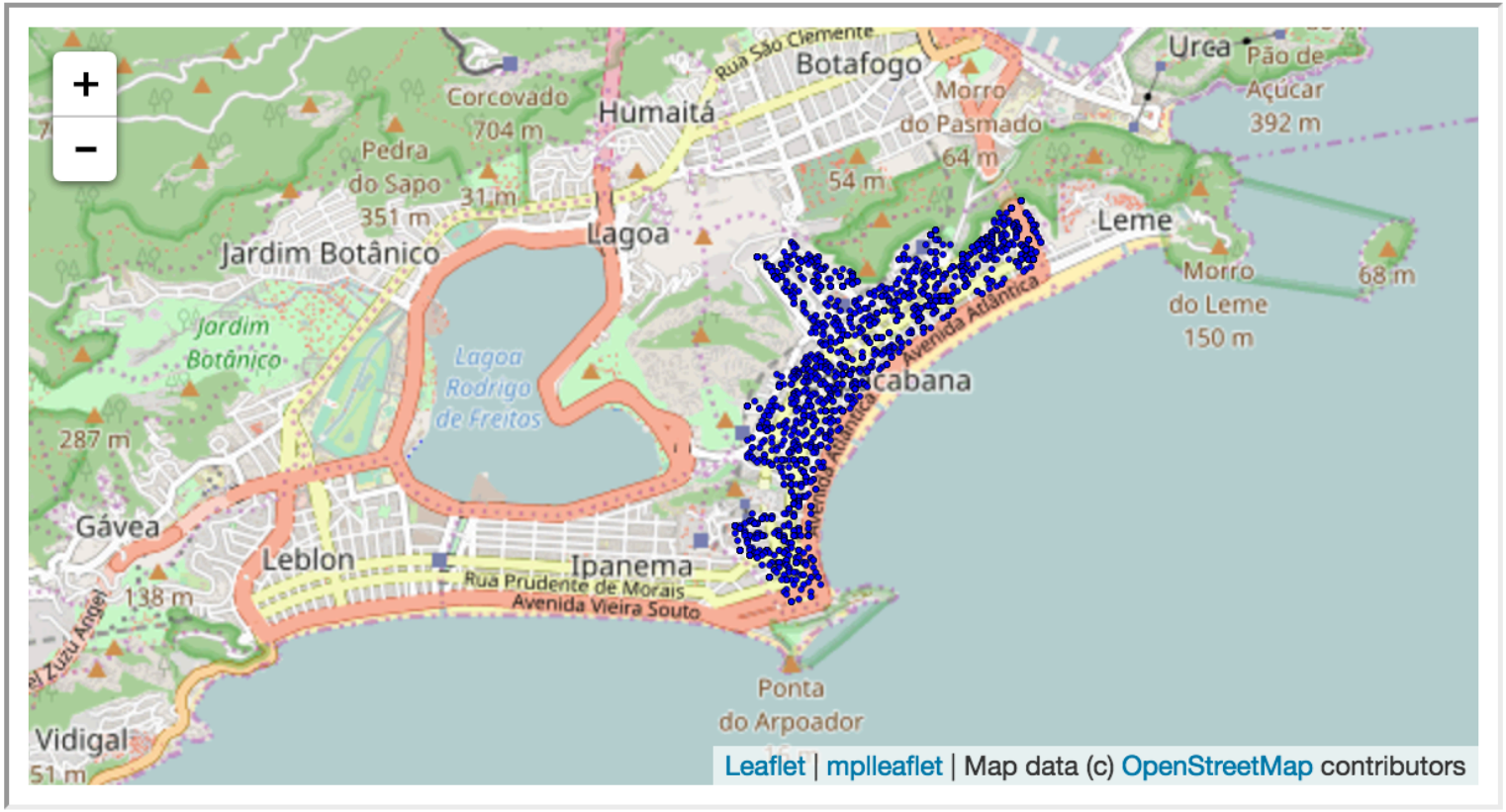
```
# Exibindo a localização dos imóveis
plt.scatter(df['lng'], df['lat'], marker='.')
```

```
<matplotlib.collections.PathCollection at 0x11430e890>
```



Exercício 136: mplleaflet

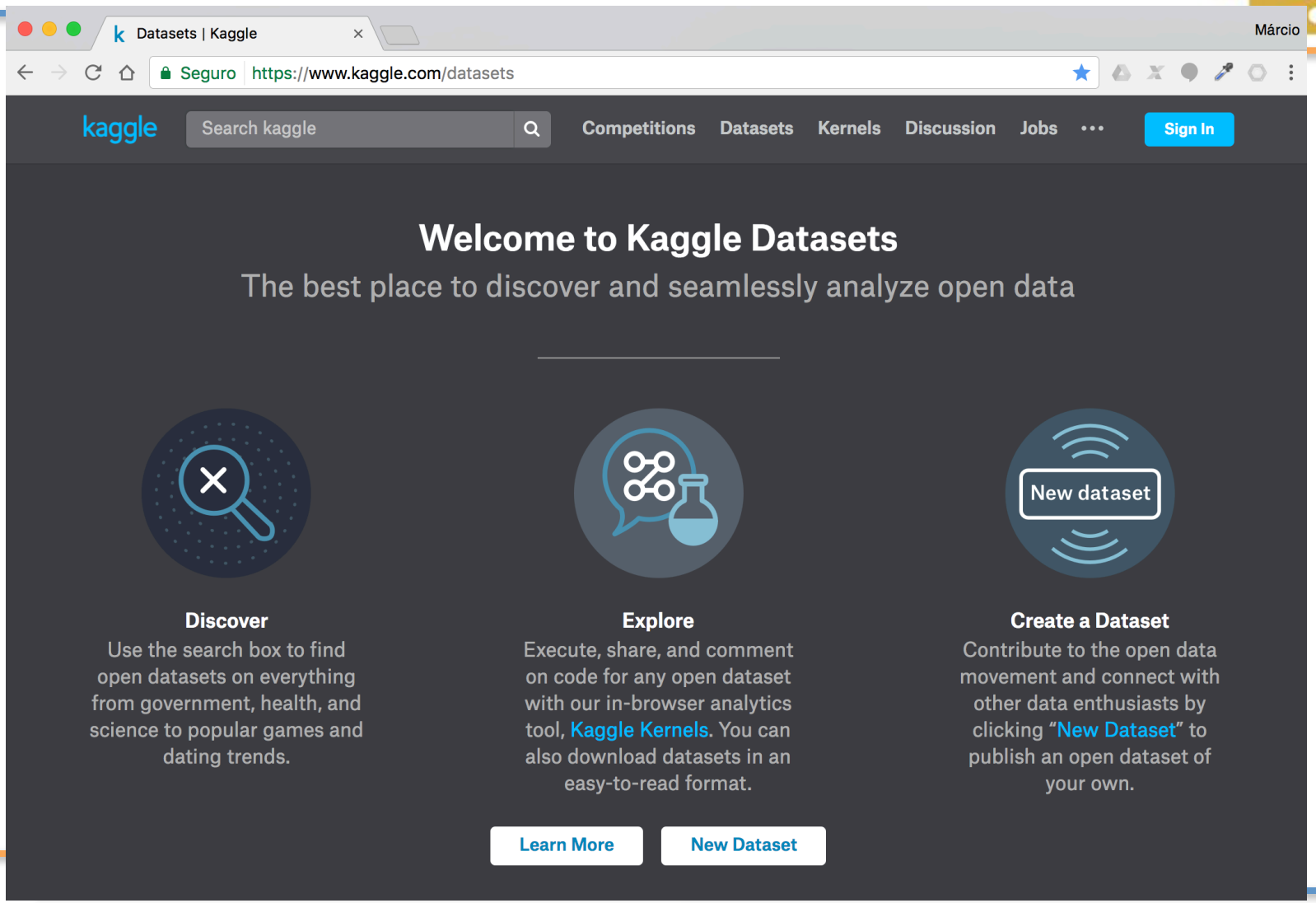
```
import mplleaflet
plt.scatter(df['lng'], df['lat'], marker='.')
mplleaflet.display()
```



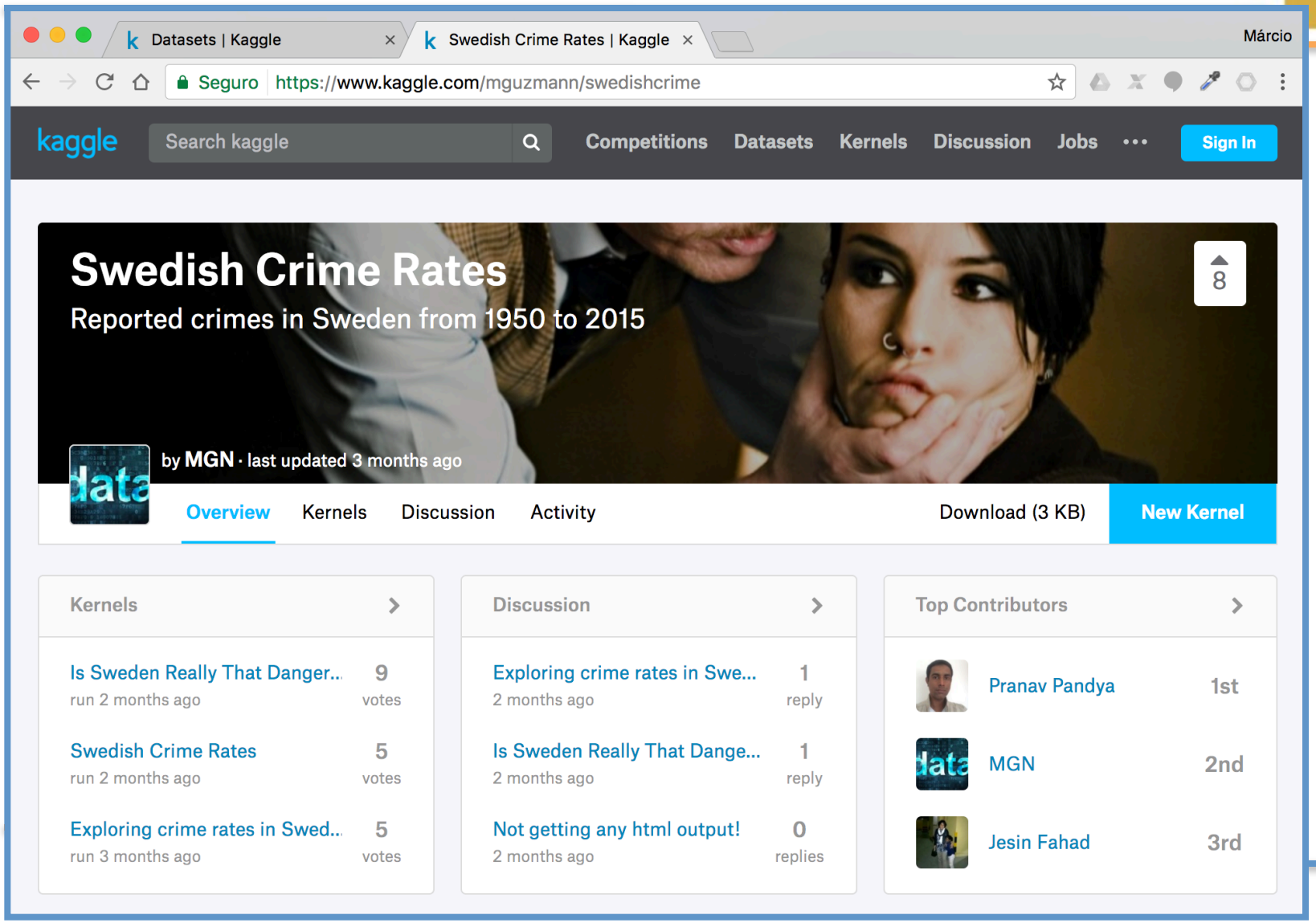
Gráficos customizados

- Agora, vamos avançar mais um pouco com os gráficos, customizando labels, legendas e colunas
- Para isso, vamos usar um novo dataset
- Site que oferece uma variedade de datasets: https://www.kaggle.com

Datasets públicos



Taxas de criminalidade sueca



Exercício 137: carga do dataset

Gráficos customizados

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Carga do dataset
df = pd.read_csv('reported.csv')
df.head()
```

	Year	crimes.total	crimes.penal.code	crimes.person	murder	assault	sexual.offenses
0	1950	2784	2306	120	1	105	40
1	1951	3284	2754	125	1	109	45
2	1952	3160	2608	119	1	104	39
3	1953	2909	2689	119	1	105	45
4	1954	3028	2791	126	1	107	41

5 rows × 21 columns

Exercício 138: missing data

```
# Temos colunas com dados faltantes
df.head()
```

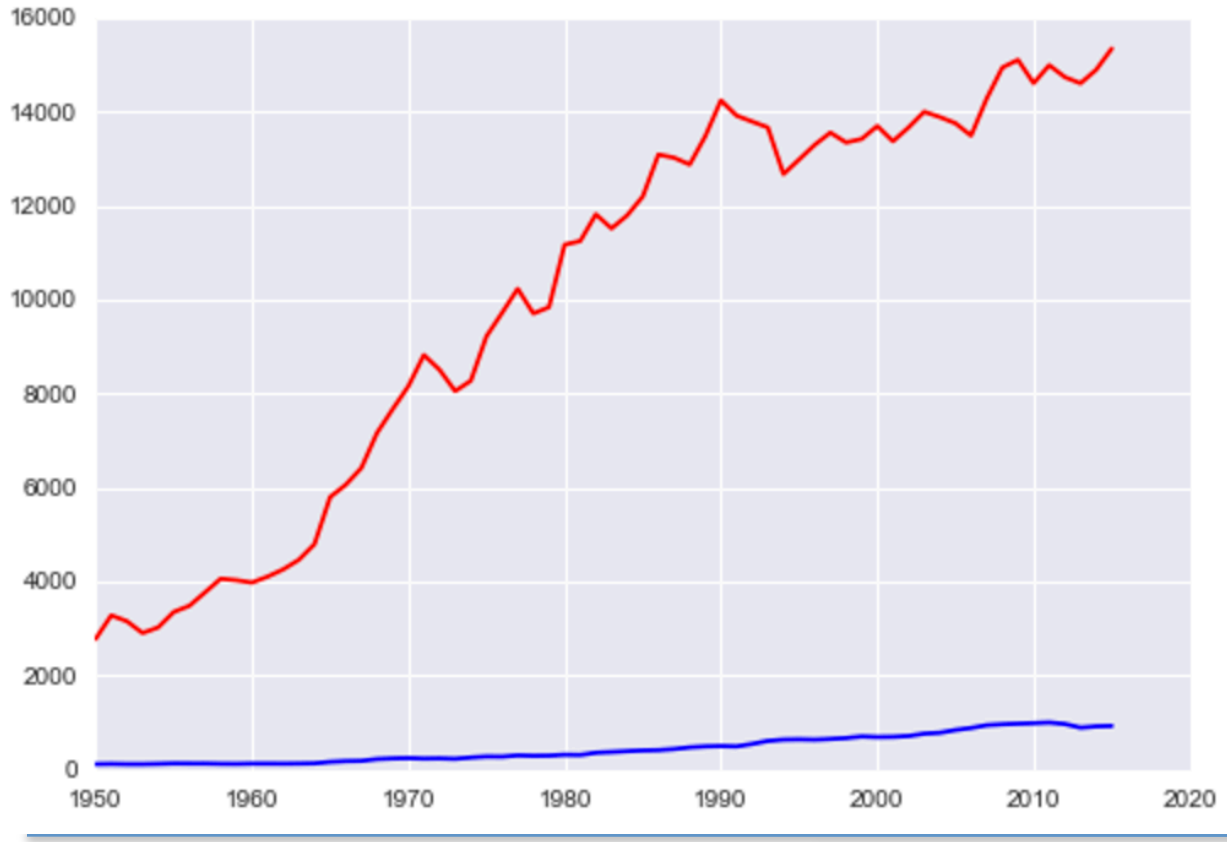
...	vehicle.theft	out.of.vehicle.theft	shop.theft	robbery	fraud	criminal.damage	other.pena
...	NaN	NaN	NaN	3	209	72	477
...	NaN	NaN	NaN	3	310	73	530
...	NaN	NaN	NaN	3	217	82	553
...	NaN	NaN	NaN	4	209	88	220
...	NaN	NaN	NaN	4	236	101	237

```
# Vamos atribuir ZERO a colunas com dados faltantes
df.fillna(0, inplace=True)
df.head()
```

...	vehicle.theft	out.of.vehicle.theft	shop.theft	robbery	fraud	criminal.damage	other.pena
...	0.0	0.0	0.0	3	209	72	477
...	0.0	0.0	0.0	3	310	73	530
...	0.0	0.0	0.0	3	217	82	553
...	0.0	0.0	0.0	4	209	88	220
...	0.0	0.0	0.0	4	236	101	237

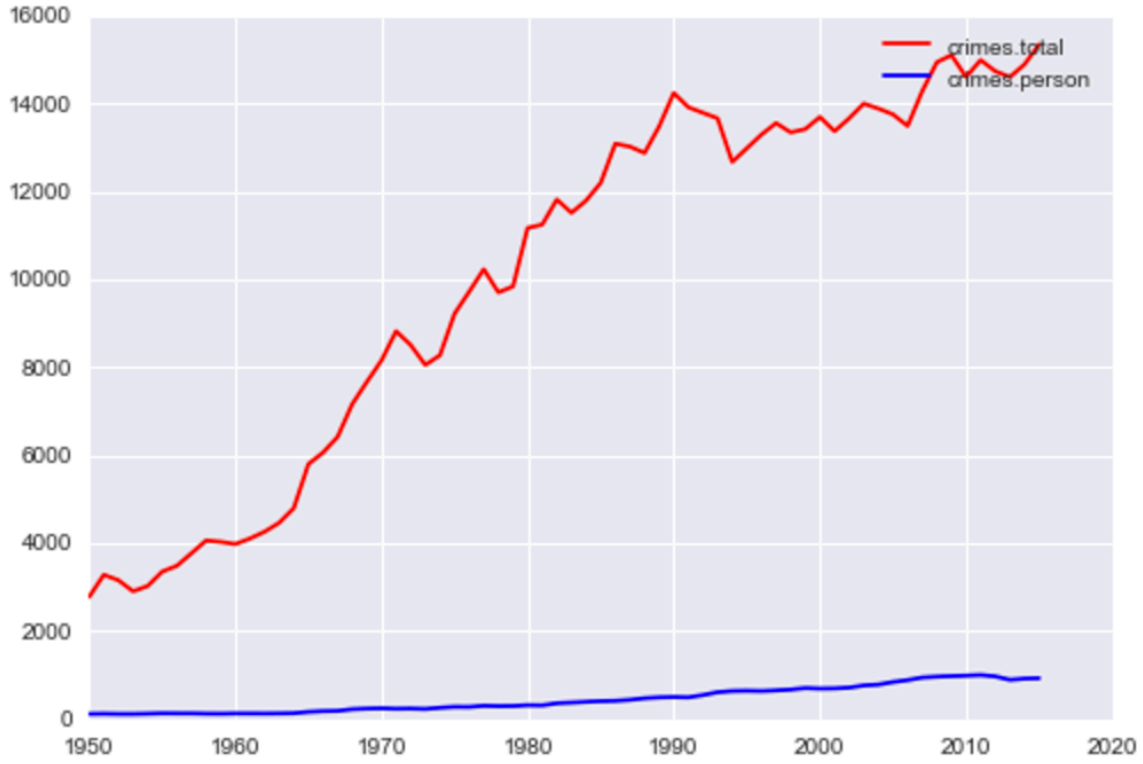
Exercício 139: pessoas e total

```
# Qual a relação entre pessoas envolvidas e o total de crimes, por ano?
plt.plot(df['Year'], df['crimes.total'], '-r')
plt.plot(df['Year'], df['crimes.person'], '-b')
plt.show()
```



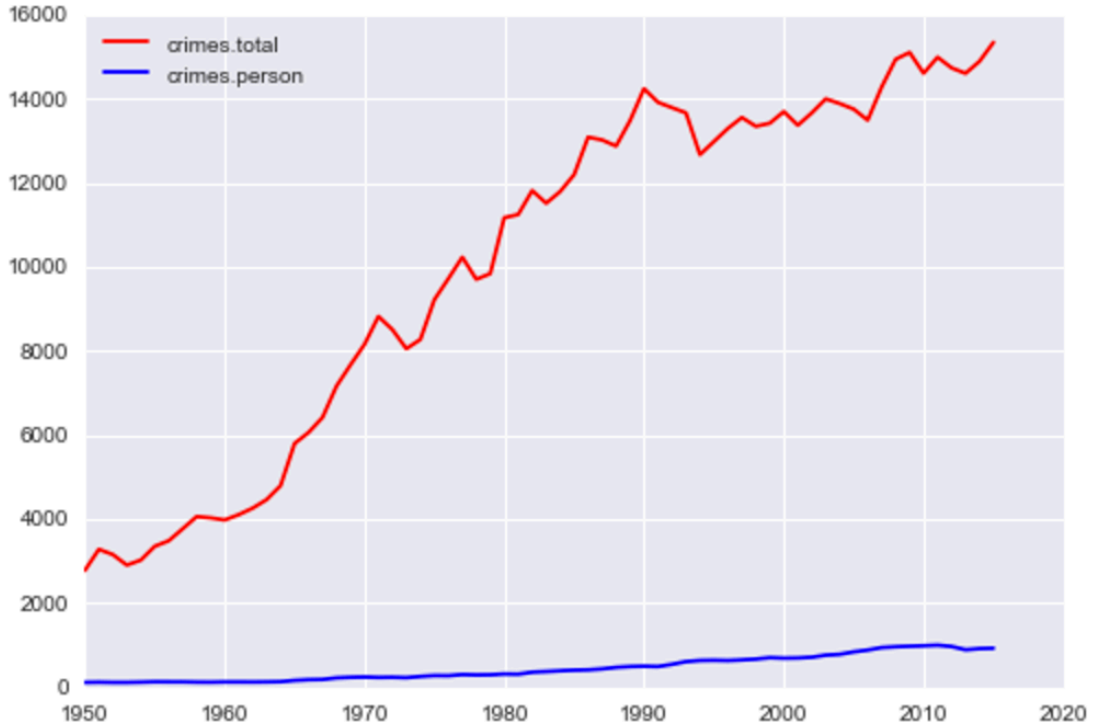
Exercício 140: ax.legend

```
# Mais controle sobre o plot - mas a legenda não ficou legal
fig, ax = plt.subplots()
ax.plot(df['Year'], df['crimes.total'], '-r')
ax.plot(df['Year'], df['crimes.person'], '-b')
ax.legend()
plt.show()
```



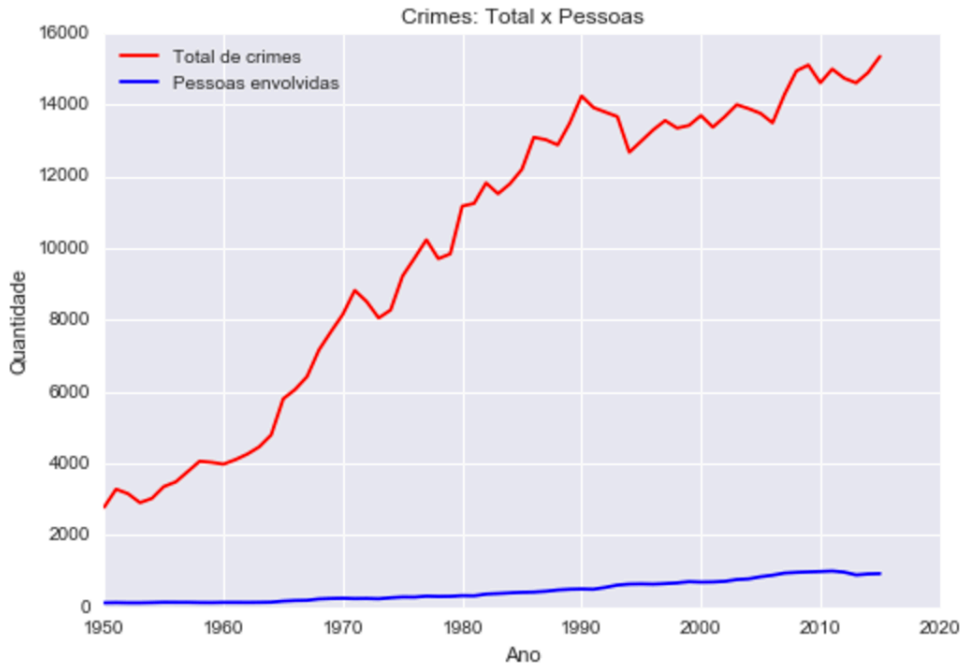
Exercício 141: loc='upper left'

```
fig, ax = plt.subplots()
ax.plot(df['Year'], df['crimes.total'], '-r')
ax.plot(df['Year'], df['crimes.person'], '-b')

# mudando o local da legenda
ax.legend(loc='upper left')

plt.show()
```



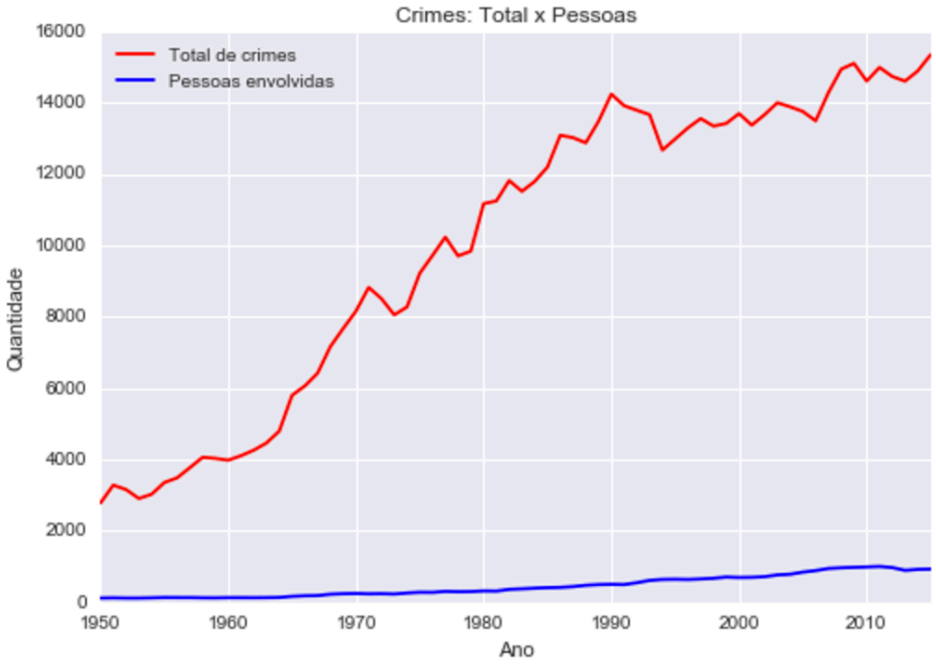
Exercício 142: legend.ylabel.xlabel

```
fig, ax = plt.subplots()
ax.plot(df['Year'], df['crimes.total'], '-r')
ax.plot(df['Year'], df['crimes.person'], '-b')

# Nome da legenda
ax.legend(['Total de crimes', 'Pessoas envolvidas'], loc='upper left')

# Label dos eixos X e Y
ax.set_xlabel('Ano')
ax.set_ylabel('Quantidade')
ax.set_title('Crimes: Total x Pessoas')

plt.show()
```



Exercício 143: set_xlim(min, max)

```
In [17]: fig, ax = plt.subplots()
ax.plot(df['Year'], df['crimes.total'], '-r')
ax.plot(df['Year'], df['crimes.person'], '-b')
ax.legend(['Total de crimes', 'Pessoas envolvidas'], loc='upper left')
ax.set_xlabel('Ano')
ax.set_ylabel('Quantidade')
ax.set_title('Crimes: Total x Pessoas')

# Limites mínimo e máximo para o eixo X e Y
ax.set_xlim(df['Year'].min(), df['Year'].max())

plt.show()
```



Exercício 144: séries temporais

Gráficos de séries temporais

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as dates
import datetime as dt
```

```
# Log da page views do site Python para Zumbies
# Segundo o google analytics
# Log de 2 meses, hora a hora
df = pd.read_csv('ppz-jan-fev-2017.csv')
df.head()
```

	hour	views
0	0	9
1	1	0
2	2	1
3	3	2
4	4	4

Exercício 145: Gráficos

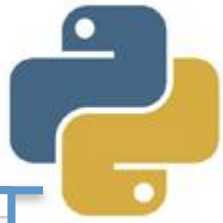
Gráficos de séries temporais

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as dates
import datetime as dt
```

```
# Log da page views do site Python para Zumbies
# Segundo o google analytics
# Log de 2 meses, hora a hora
df = pd.read_csv('ppz-jan-fev-2017.csv')
df.head()
```

	hour	views
0	0	9
1	1	0
2	2	1
3	3	2
4	4	4

Exercício 146: hora para data



```
def to_date(value):
    ''' Função para convertar as horas coletadas em datetime
    Args:
        value (int): Hora a ser convertida.
                     A contagem inicia em 01/01/2017 00:00

    Returns:
        datetime: Hora convertida em Datatime
    '''
    return dt.datetime(2017, 1, 1) + dt.timedelta(hours=value)
```

```
# Criação da coluna 'date' a partir de to_date(hora)
df['date'] = df['hour'].apply(to_date)
df.head()
```

	hour	views	date
0	0	9	2017-01-01 00:00:00
1	1	0	2017-01-01 01:00:00
2	2	1	2017-01-01 02:00:00
3	3	2	2017-01-01 03:00:00
4	4	4	2017-01-01 04:00:00

Exercício 147: delete 'hour'

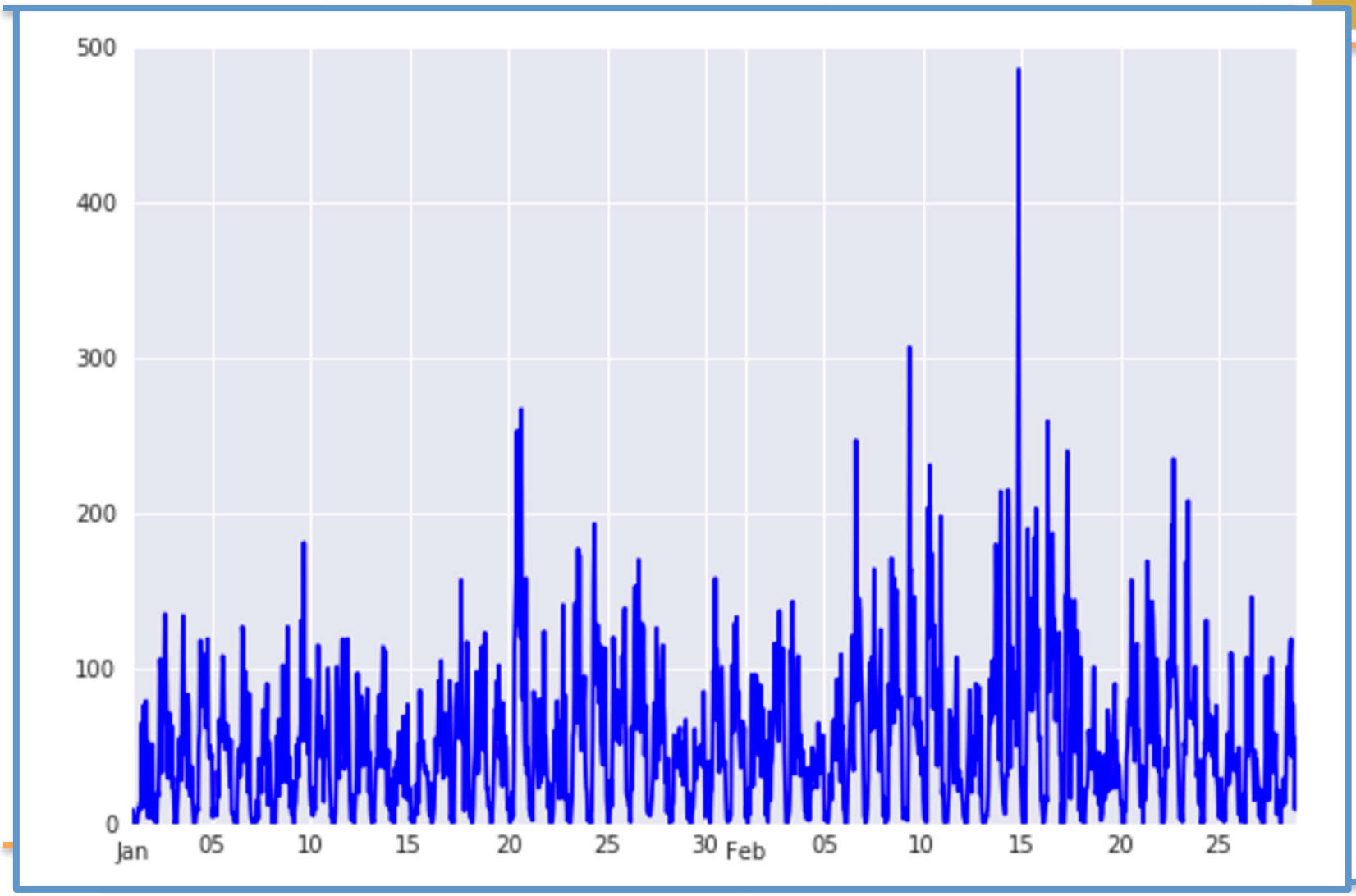
```
# Remove a coluna de hora
del df['hour']

#Atualiza o índice - Define melhor a série temporal para o pandas
df.set_index(['date'], inplace=True)
df.head()
```

	views
date	
2017-01-01 00:00:00	9
2017-01-01 01:00:00	0
2017-01-01 02:00:00	1
2017-01-01 03:00:00	2
2017-01-01 04:00:00	4

Exercício 148: plot da série

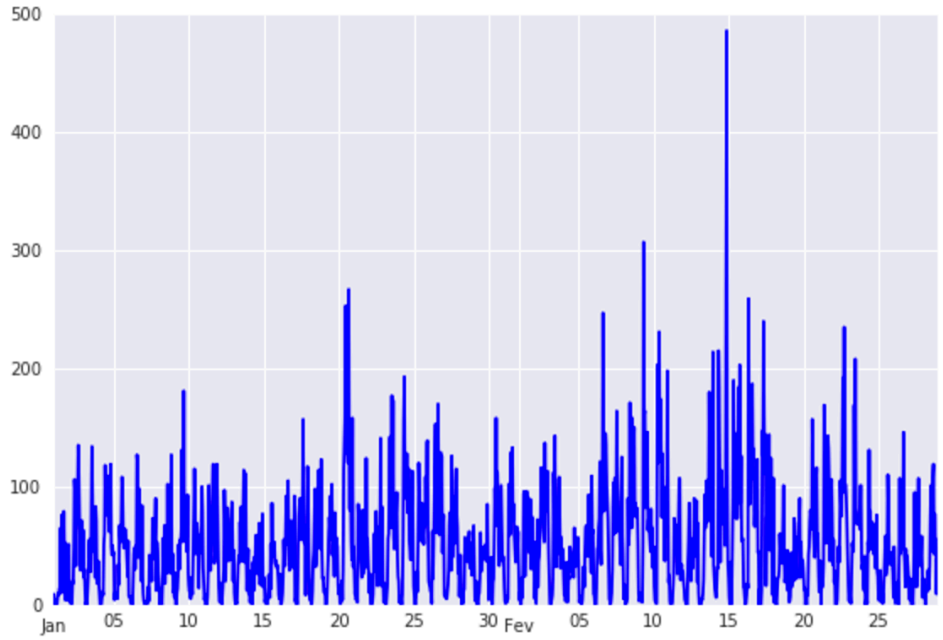
```
# Plotar a série
fig, ax = plt.subplots()
# Converter o índice para o valor de datas do python
ax.plot_date(df.index.to_pydatetime(), df['views'], 'b-')

# Locator é a forma de marcação do eixo X
# bymonthday=range(5, 32, 5) - Exibir de 5 a 31, passo 5
ax.xaxis.set_minor_locator(dates.DayLocator(bymonthday=range(5, 32, 5)))
# formatação do label
ax.xaxis.set_minor_formatter(dates.DateFormatter('%d'))

# Exibir linhas de corte, a cada 5 dias
ax.xaxis.grid(True, which='minor')
# Exibir o grid normal
ax.yaxis.grid(True)

#Marcação de mês
ax.xaxis.set_major_locator(dates.MonthLocator())
# Mês com 3 caracteres
ax.xaxis.set_major_formatter(dates.DateFormatter('%b'))

#função de ajuste do gráfico
plt.tight_layout()
plt.show()
```

Resultado



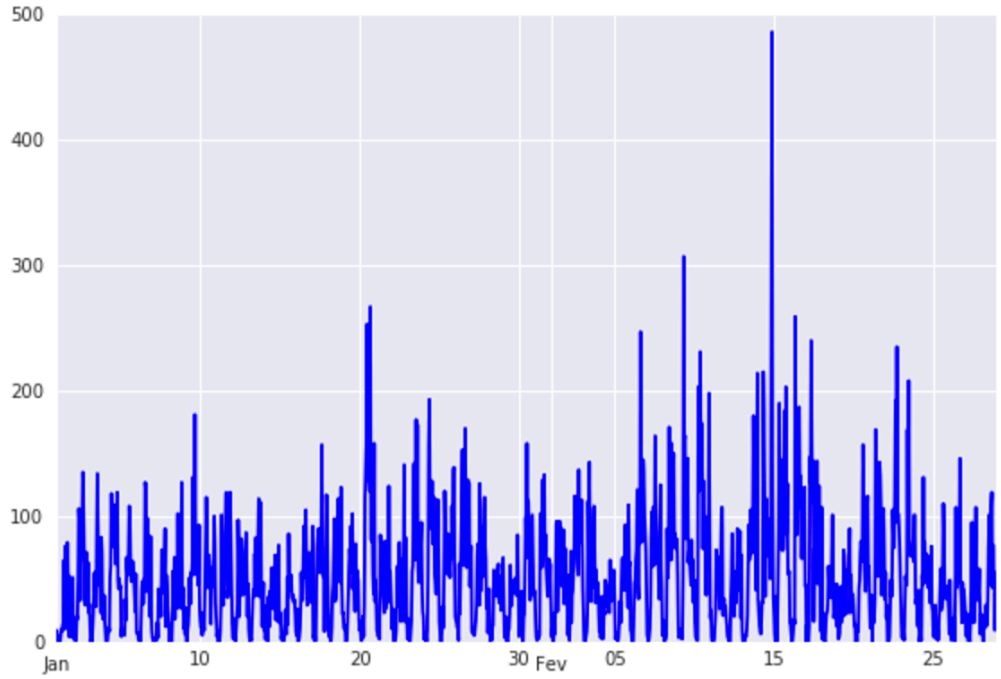
Exercício 149: pt_BR

```
import locale
locale.setlocale(locale.LC_ALL, 'pt_BR')

fig, ax = plt.subplots()
ax.plot_date(df.index.to_pydatetime(), df['views'], 'b-')
ax.xaxis.set_minor_locator(dates.DayLocator(bymonthday=range(5, 32, 5)))
ax.xaxis.set_minor_formatter(dates.DateFormatter('%d'))
ax.xaxis.grid(True, which='minor')
ax.yaxis.grid(True)
ax.xaxis.set_major_locator(dates.MonthLocator())
ax.xaxis.set_major_formatter(dates.DateFormatter('%b'))
plt.tight_layout()
plt.show()
```



Exercício 150: interval=2

```
fig, ax = plt.subplots()
ax.plot_date(df.index.to_pydatetime(), df['views'], 'b-')

# Só exibe na 2a ocorrência
ax.xaxis.set_minor_locator(dates.DayLocator(bymonthday=range(5, 32, 5), interval=2))

ax.xaxis.set_minor_formatter(dates.DateFormatter('%d'))
ax.xaxis.grid(True, which='minor')
ax.yaxis.grid(True)
ax.xaxis.set_major_locator(dates.MonthLocator())
ax.xaxis.set_major_formatter(dates.DateFormatter('%b'))
plt.tight_layout()
plt.show()
```



FIM

Contatos

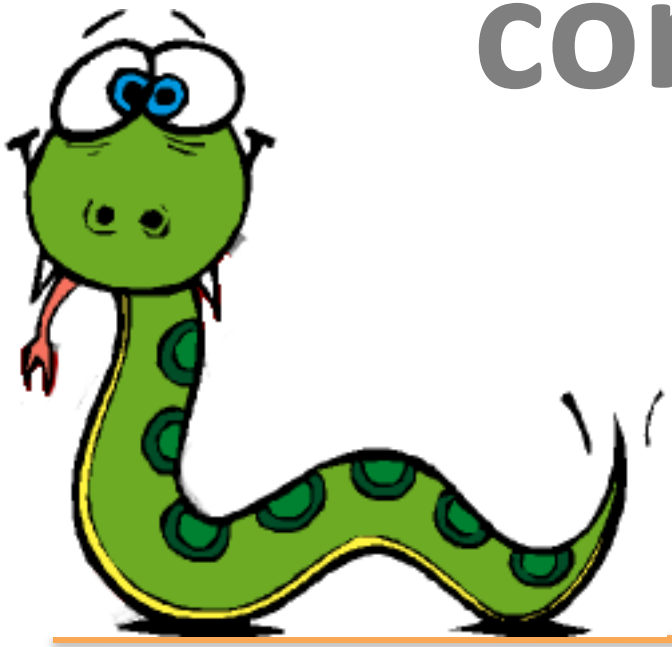
- Márcio Palheta
  - marcio.palheta@gmail.com
  - @marciopalheta
  - https://sites.google.com/site/marciopalheta/

Bibliografia

- LIVRO: Apress - Beginning Python From Novice to Professional
- LIVRO: O'Relly - Learning Python
- http://www.python.org
- http://www.python.org.br

- Mais exercícios:
  - http://wiki.python.org.br/ListaDeExercicios

- Documentação do python:
  - https://docs.python.org/2/



Análise de dados com Pandas e Seaborn

Márcio Palheta, M.Sc.
marcio.palheta@gmail.com