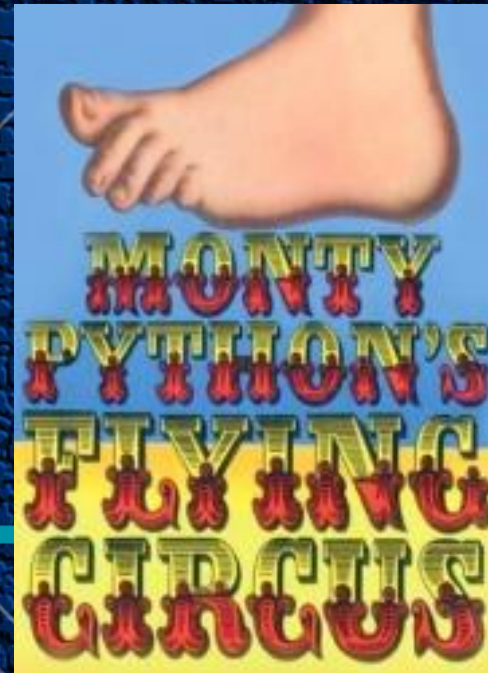


Aula 01

Linguagem de Programação Estatística: Python
MBA em Data Science e Machine Learning - UNIP
Prof. M.e Victor de Assis Rodrigues

AGENDA

Por que
utilizar
Python?



Características
da
linguagem

Numpy
e
Pandas

Séries
Temporais

Livros

Data Science Fundamentals for Python and MongoDB

— David Paper

apress

Making Everything Easier!

Python for Data Science FOR DUMMIES[®]

A Wiley Brand

Learn to:

- Take advantage of Python data analysis programming
- Work with Python objects, functions, modules, and libraries
- Apply statistical concepts such as probability and random distributions
- Use NumPy, SciPy, Scikit-learn, and Pandas libraries

John Paul Mueller
Luca Massaron



Agile Tools for Real-World Data

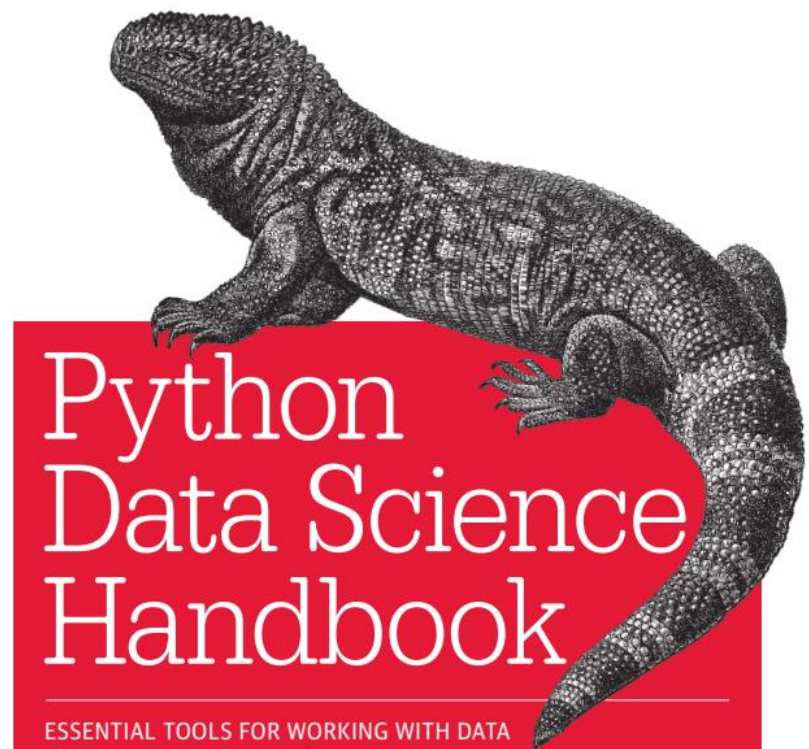
Python for Data Analysis



O'REILLY[®]

Wes McKinney

O'REILLY[®]



powered by



Jake VanderPlas

www.allitebooks.com

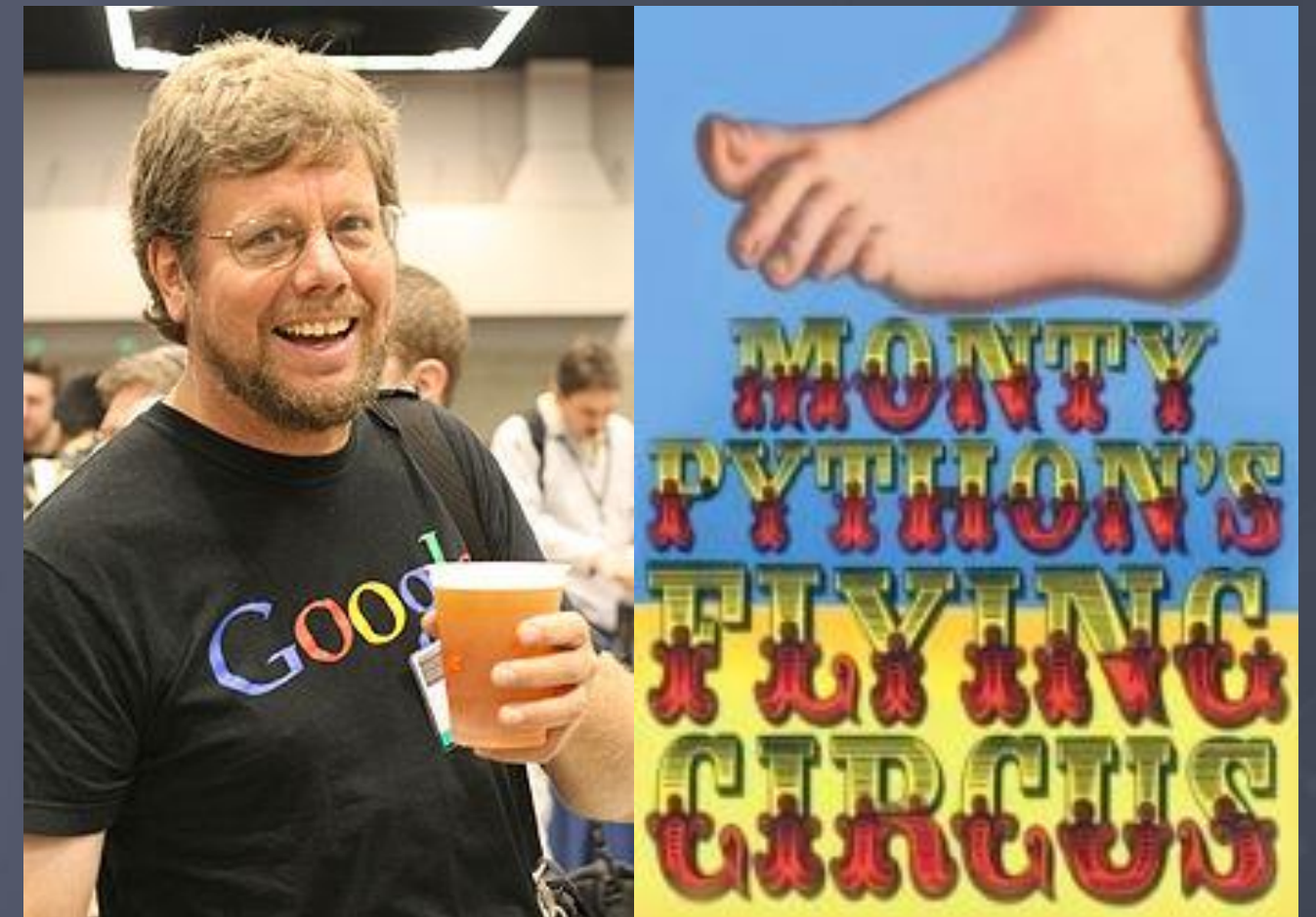
História do Python

Surgimento no final dos anos 80;

Idealizado por Guido Van Rossum em 1982;

Percebi que o desenvolvimento de utilitários para administração de sistema em C (do Amoeba) estava tomando muito tempo. Além disso, fazê-los em shell Bourne não funcionaria por diversas razões. [...] Portanto, havia necessidade de uma linguagem que "preencheria o vazio entre C e o shell [...]"

—Guido Van Rossum



Nome inspirado no seriado britânico “Monty Python’s Flying Circus”.

<https://www.youtube.com/watch?v=2YCmIKS-8RY>

Por que estudar Python?

Python é uma linguagem muito interessante quando trabalhamos com conjuntos de dados;

Não foi projetada para se trabalhar com data analysis, **e sim** tarefas relacionadas a computação científica;

Por ser uma linguagem versátil, a própria comunidade acelerou sua popularização;



































Grande crescimento quando surgiram:

Django / Pyramid (framework p/ aplicações Web)

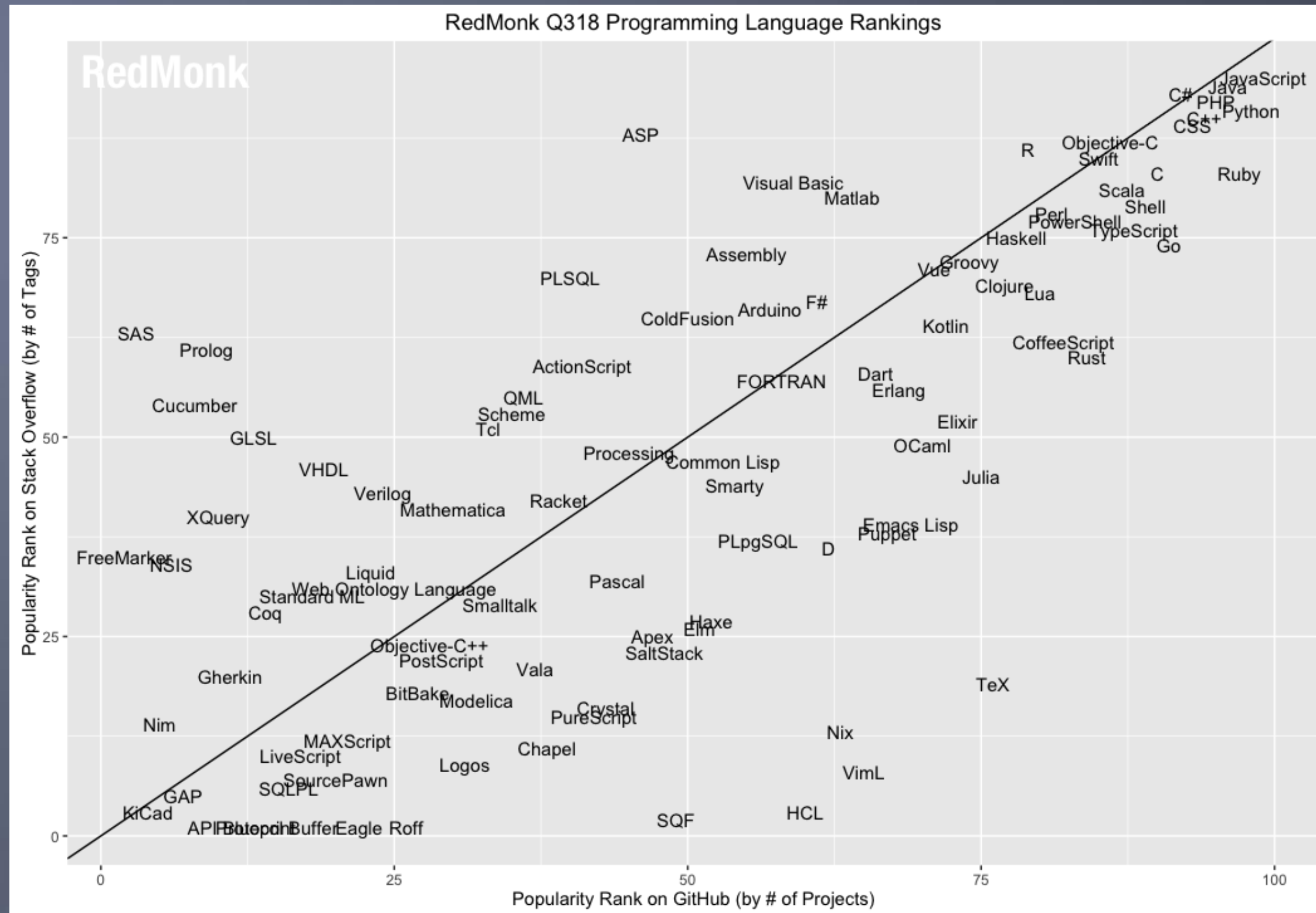
Bibliotecas NumPy, Matplotlib e Scikit-learn (voltado para Ciência de Dados)

Por que estudar Python?

Ranking da IEEE Spectrum (2018)

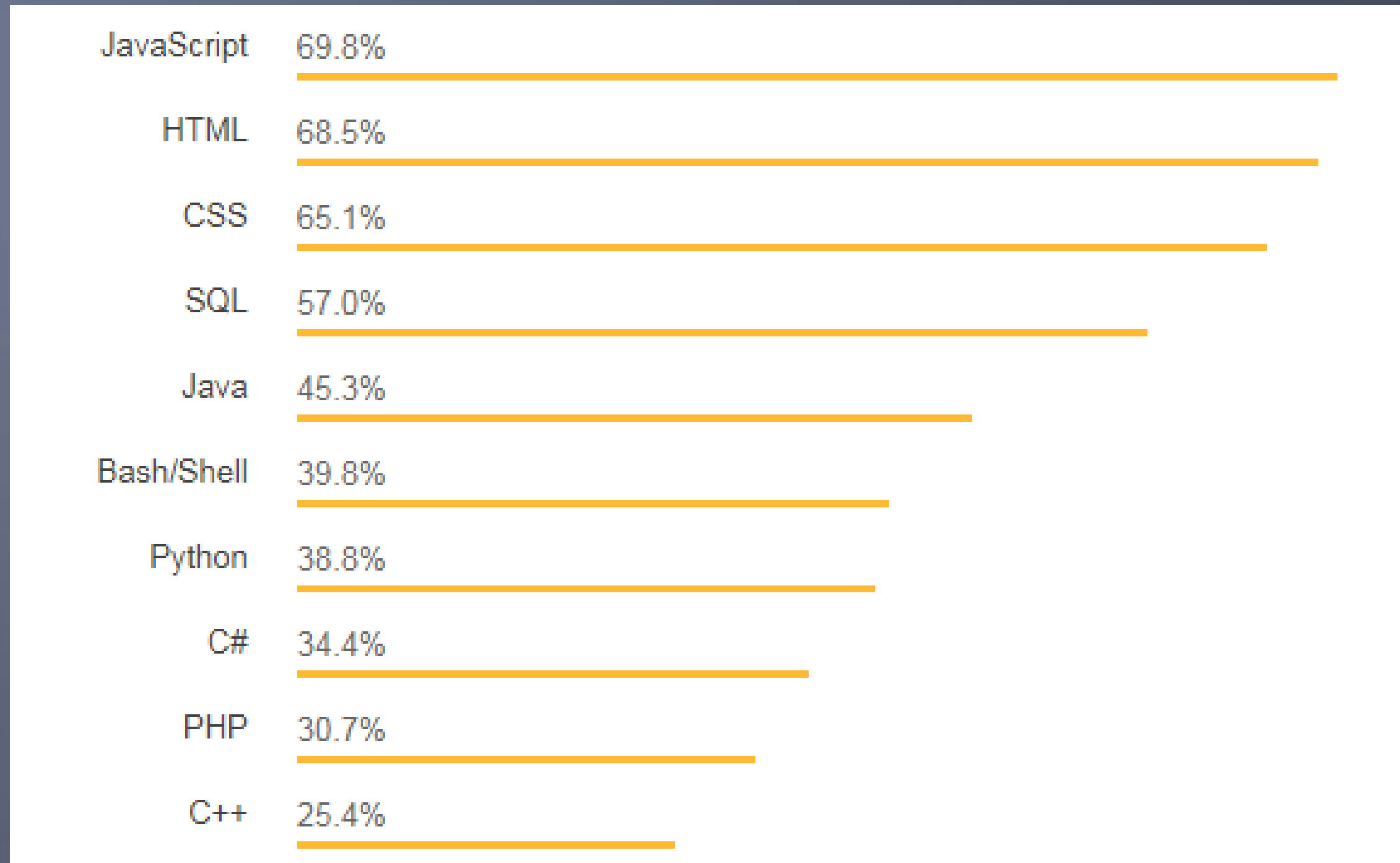
Language Rank	Types	Spectrum Ranking
1. Python	  	100.0
2. C++	  	99.7
3. Java	  	97.5
4. C	  	96.7
5. C#	  	89.4
6. PHP		84.9
7. R		82.9
8. JavaScript	 	82.6
9. Go	 	76.4
10. Assembly		74.1
11. Matlab		72.8
12. Scala	 	72.1
13. Ruby	 	71.4
14. HTML		71.2
15. Arduino		69.0
16. Shell		66.1
17. Perl	 	57.4
18. Swift	 	53.9

Por que estudar Python?



Ranking da Redmonk (Jun/2018)

Por que estudar Python?



Ranking de popularidade – Stackoverflow (2018)

Por que estudar Python?

Características da linguagem que se destacam:

- Modularização de código;
- Sistemas Embarcados;
- Bibliotecas p/ praticamente todos os tipos de problemas;
- Comunidade “super” ativa e colaborativa;
- Fácil de aprender;
- Fácil de instalar e configurar;
- Sua utilização facilita o entendimento de conceitos como Lógica de Programação e OOP.

Passo a passo de um sistema p/ Análise de Dados

Interação

- Ler e escrever em uma variedade de arquivos e bancos de dados

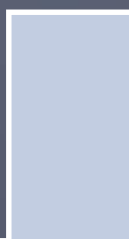


Preparação

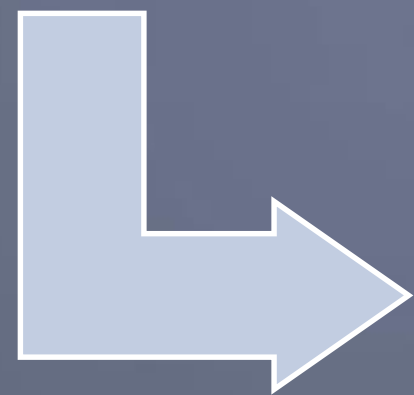
- Limpezas, refatoração, *munging*, combinações de dados, reshaping, slicing etc.



Transformação

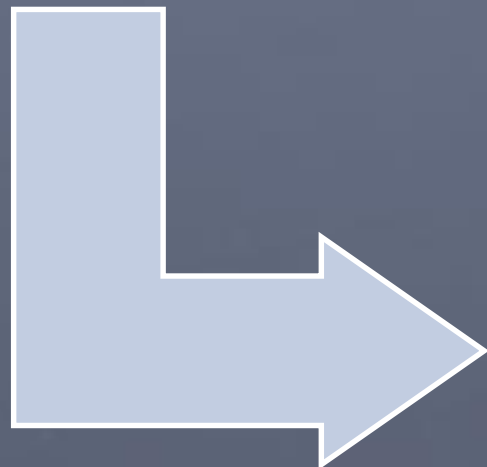
- Aplicação de operações matemáticas e estatísticas, buscando obter, analisar e validar informações
- 

Passo a passo de um sistema p/ Análise de Dados



Modelagem

- Dados conectados a uma função ou modelo estatístico, aprendizado de máquina ou outras ferramentas computacionais



Apresentação

- Gráficos interativos, visualização de grafos e/ou sumários (tabelas)

Bibliotecas Principais

Lista das bibliotecas principais para Data Science e Machine Learning: (BOBRIAKOV, 2018)

Estatística:

Numpy

SciPy

Pandas

StatsModels

Bibliotecas Principais

Lista das bibliotecas principais para Data Science e Machine Learning: (BOBRIAKOV, 2018)

Machine Learning:

Scikit-learn

XGBoost/LightGBM/CatBoost

Eli5

Deep Learning:

Keras

TensorFlow

PyTorch

Bibliotecas Principais

Lista das bibliotecas principais para Data Science e Machine Learning: (BOBRIAKOV, 2018)

Visualização:

Matplotlib

Seaborn

Plotly

Pydot

Outras:

NLTK (proc. Linguagem natural)

OpenCV* (proc. Imagens e visão comp.)

mlPy* (*machine learning*)

Theano* (algumas alternativas p/ o Numpy)

* não citado no artigo

Numpy

Utilizamos o Numpy para trabalhar com grandes volumes de dados.

Ele fornece:

- ndarray: uma estrutura de array multidimensional que facilita operações aritméticas e binárias (broadcasting);
- Funções prontas que economizam a escrita de ifs e Loops;
- Leitura e escritas de arquivos;
- Álgebra linear, processos combinatórios, Transformada de Fourier;
- Integração com C, C++ e Fortran e vários banco de dados;

Pandas

Construída sobre o **Numpy** (biblioteca *on top*, como diversas outras);

Tem seu foco em fornecer **funções** comumente utilizadas para **finanças, ciências sociais e engenharias**.

Porém, sua principal vantagem é o fornecimento de **estruturas próprias** com suporte automático ou explícito ao refinamento de dados;

Suporte dados temporais e não temporais **numa mesma estrutura**;

Tratamento aos dados como se fosse **operações comuns** em um banco de dados.

Matplotlib

Ferramenta padrão para criação de gráficos;

Possui diversos tipos de gráficos, diversas configurações;

Possíveis plotar diversos gráficos de forma separada (lado-a-lado) ou realizando um merge das informações.



****Sugestão de leitura:** [Link](#)

IPython

É um conjunto de subprojetos com objetivo de facilitar o desenvolvimento de aplicações, **principalmente** na linguagem Python;

Com o tempo, esses subprojetos **foram movidos** e hoje fazem parte do Project Jupyter.

As duas distribuições ainda existem, porém após o lançamento do Python 4.x **permanecerá apenas** o Project Jupyter.

SciPy

Coleção de pacotes que são utilizados em diferentes problemas da computação científica.

Alguns pacotes:

`scipy.integrate`: diversas equações matemáticas

`scipy.linalg`: álgebra linear e decomposição

`scipy.optimize`: otimizadores, algoritmos de busca, minimizadores

`scipy.signal`: processamento de sinais (filtragens, thresholding, etc)

`scipy.stats`: distribuições contínuas e discretas

Diferenças entre o Python 2.7 e 3.x

Principal ponto: o Python 2.7 será descontinuado em 2020;

Existem algumas mudanças na chamada de funções, ver mais detalhes nesse [link](#);

Código Python 3 não é retrocompatível;

Podem surgir problemas na utilização de bibliotecas. Ficar atento as versões!

Problemas com a licença de algoritmos do OpenCV (abrangem apenas a versão 2.7)

Editor de Texto **OU** IDE???



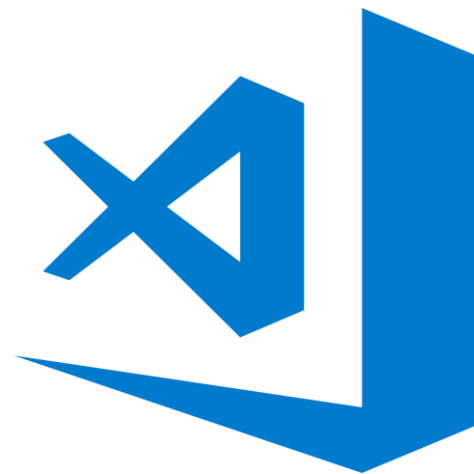
Jupyter



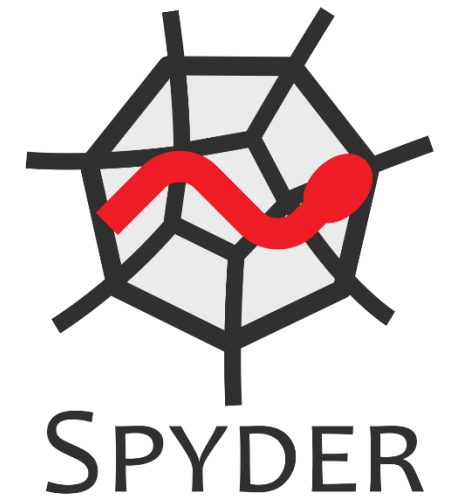
Atom



Sublime



VS Code



Pycharm

Primeira Aplicação

1. Abrir o Anaconda Prompt;
2. Comando: `python`
3. Comando: `print("Oi Mundo")`

Primeira Aplicação com o Jupyter Notebook/Lab

1. Abrir o Anaconda Prompt;
2. Mudar o diretório do CMD para uma pasta de trabalho (comando:
cd nomeDoDiretorio)
3. Digitar Jupyter Notebook

```
(base) C:\Users\Victor>d:
```

```
(base) D:\>cd MBADataScience
```

```
(base) D:\MBADataScience>jupyter notebook
```

Aspectos Básicos da linguagem

- IF, FOR, WHILE...
- Array / Vetores / Tuplas;
- Funções;
- Classes;
- Slicing;
- Indexing com Slice;
- Operações entre Arrays;

- Fonte: <http://github.com/dsacademybr>

Relembrando os conceitos principais:

- Slicing: Técnica de fatiamento (subdivisão) de dados;
- Listas: São elementos mutáveis, utilizamos [] na declaração;
- Dicionário: São elementos mutáveis, utilizamos {} na declaração e acessamos seus valores por um rótulo;
- Tupla: São elementos imutáveis, utilizamos ()

Início com Pandas

Como nós vimos, o Numpy providencia uma estrutura chamada ndarray:

- Facilita a organização de dados;
- Útil para um bom número de tarefas;
- Ineficiente quando queremos classificar grupo de dados (colocar labels, minimizar perda de dados, mapeamentos, etc).

O Pandas fornece diversos recursos para essas necessidades.

Séries

Séries

É um vetor (array unidimensional) indexado de dados. Podemos criá-los e utilizá-los como uma lista:

```
In[2]: data = pd.Series([0.25, 0.5, 0.75, 1.0])
```

```
data
Out[2]: 0    0.25
        1    0.50
        2    0.75
        3    1.00
        dtype: float64
```

```
In[3]: data.values
```

```
Out[3]: array([ 0.25,  0.5 ,  0.75,  1.  ])
```

```
In[6]: data[1:3]
```

```
Out[6]: 1    0.50
        2    0.75
        dtype: float64
```

Séries

Séries com Dicionários

Recurso para relacionar um valor da série com uma descrição (label).

```
populacao = {'Bauru': 337094,  
             'Lençóis Paulista': 66664,  
             'Marília': 197342,  
             'Lins': 77021,  
             'Bocaina': 11810}
```

```
populacao = pd.Series(populacao)
```

```
populacao['Lins']
```

```
Out[15]: 77021
```

```
populacao['Lençóis Paulista':'Lins']
```

```
Out[16]: Lençóis Paulista    66664  
         Marília           197342  
         Lins              77021  
         dtype: int64
```

Séries

Séries com Dicionários

Criação - Valores separados dos labels

```
import numpy as np
import pandas as pd

v = [0, 1, 2, 3]
l = ['a', 'b', 'c', 'f']

m = pd.Series(v, l)
print(m)
```

```
a    0
b    1
c    2
f    3
dtype: int64
```


Dataframes

Podem ser classificados como arrays multidimensionais com indexação, e que permitem o rótulo de dados.

```
In [20]: populacao = {'Bauru': 337094,
                        'Lençóis Paulista': 66664,
                        'Marília': 197342,
                        'Lins': 77021,
                        'Bocaina': 11810}

populacao = pd.Series(populacao)

postosGasolina = [100, 30, 80, 25, 20 ]

mapa = pd.DataFrame(
    {
        'populacao': populacao,
        'postos': postosGasolina
    }
)
mapa
```

Out[20]:

	populacao	postos
Bauru	337094	100
Lençóis Paulista	66664	30
Marília	197342	80
Lins	77021	25
Bocaina	11810	20

Dataframes

Quando queremos pegar os “nomes” das linhas utilizamos o atributo *index*.

```
mapa.index
```

```
Index(['Bauru', 'Lençois Paulista', 'Marilia', 'Lins', 'Bocaina'], dtype='object')
```

Quando queremos pegar os “nomes” das colunas, utilizamos o atributo *columns*.

```
mapa.columns
```

```
Index(['populacao', 'postos'], dtype='object')
```

Dataframes

Temos a função `describe()` que mostra estatísticas básicas do Dataframe

```
mapa.describe()
```

	populacao	postos
count	5.000000	5.000000
mean	137986.200000	51.000000
std	130279.57657	36.469165
min	11810.000000	20.000000
25%	66664.000000	25.000000
50%	77021.000000	30.000000
75%	197342.000000	80.000000
max	337094.000000	100.000000

Dataframes

Para adicionar uma nova coluna, basta passar o nome da coluna como índice do Dataframe e atribuir a lista de valores desejadas.

```
In [45]: lojas = [450, 45, 120, 32, 10 ]  
  
mapa['lojas'] = lojas  
mapa
```

	populacao	postos	lojas
Bauru	337094	100	450
Lençóis Paulista	66664	30	45
Marília	197342	80	120
Lins	77021	25	32
Bocaina	11810	20	10

Para excluir usamos a função drop()

```
mapa = mapa.drop(columns='postos')  
mapa
```

	populacao	lojas
Bauru	337094	450
Lençóis Paulista	66664	45
Marília	197342	120
Lins	77021	32
Bocaina	11810	10

Seleção e agregação de dados

Para aprendermos melhor sobre os comandos de seleção e agregação, iremos realizar a leitura do arquivo “Season_Stats.csv”, disponível no Kaggle (nesse [link](#)).

Para realizar a leitura, vamos executar o comando abaixo:

```
import pandas as pd
dados = pd.read_csv('Seasons_Stats.csv')
```

Seleção e agregação de dados

Algumas operações:

```
dados.count() # Qtde de dados de cada coluna
```

```
dados['Age'].mean() # Média de idade
```

```
dados['G'].median() # Mediana do número de jogos dos jogadores
```

```
dados['G'].describe()
```


Loc e iloc

Loc

Utilizamos para pegar os registros de um Dataframe pelo seu índice:

```
dados.loc[[1, 2, 5]]
```

	Unnamed: 0	Year	Player	Pos	Age	Tm	G	GS	MP	PER	...	FT%
1	1	1950.0	Cliff Barker	SG	29.0	INO	49.0	NaN	NaN	NaN	...	0.708
2	2	1950.0	Leo Barnhorst	SF	25.0	CHS	67.0	NaN	NaN	NaN	...	0.698
5	5	1950.0	Ed Bartels	F	24.0	NYK	2.0	NaN	NaN	NaN	...	0.667

3 rows × 53 columns

Loc e iloc

Loc[index]

Se quisermos procurar por um jogador específico, precisamos setar a coluna Player como index antes de executarmos o loc.

```
# setando a coluna jogador como índice
dados = dados.set_index('Player')
```

```
#Localizar 'Lou Williams'
dados.loc['Lou Williams']
```

	Unnamed: 0	Year	Pos	Age	Tm	G	GS	MP	PER	TS%	...	FT%
Player												
Lou Williams	18211	2006.0	PG	19.0	PHI	30.0	0.0	145.0	9.0	0.485	...	0.615
Lou Williams	18728	2007.0	PG	20.0	PHI	61.0	0.0	688.0	14.6	0.521	...	0.696
Lou Williams	19313	2008.0	PG	21.0	PHI	80.0	0.0	1862.0	16.7	0.523	...	0.783
Lou Williams	19902	2009.0	SG	22.0	PHI	81.0	0.0	1919.0	16.3	0.513	...	0.790

Loc e iloc

Loc[index, columns]

Podemos também passar por parâmetro quais colunas utilizar.

```
In [54]: dados.loc[['Lou Williams'], ['Year', 'Age', 'Pos']]
```

```
Out[54]:
```

	Year	Age	Pos
Player			
Lou Williams	2006.0	19.0	PG
Lou Williams	2007.0	20.0	PG
Lou Williams	2008.0	21.0	PG
Lou Williams	2009.0	22.0	SG

Loc e iloc

iloc

O iloc é um pouco mais simples, ele sempre irá utilizar o índice como inteiro.

```
dados.iloc[1:100:2]
```

	Unnamed: 0	Year	Pos	Age	Tm	G	GS	MP	PER	TS%
Player										
Cliff Barker	1	1950.0	SG	29.0	INO	49.0	NaN	NaN	NaN	0.435
Ed Bartels	3	1950.0	F	24.0	TOT	15.0	NaN	NaN	NaN	0.312
Ed Bartels	5	1950.0	F	24.0	NYK	2.0	NaN	NaN	NaN	0.376
Gene Berce	7	1950.0	G-F	23.0	TRI	3.0	NaN	NaN	NaN	0.275
Charlie Black	9	1950.0	F-C	28.0	FTW	36.0	NaN	NaN	NaN	0.362

Filtros

Podemos filtrar valores informando uma condição booleana dentro dos []

dados[dados.Year == 2017]																	
Unnamed: 0		Year	Pos	Age	Tm	G	GS	MP	PER	TS%	...	FT%	ORB	DRB	TRB	AST	S
Player																	
Alex Abrines	24096	2017.0	SG	23.0	OKC	68.0	6.0	1055.0	10.1	0.560	...	0.898	18.0	68.0	86.0	40.0	30
Quincy Acy	24097	2017.0	PF	26.0	TOT	38.0	1.0	558.0	11.8	0.565	...	0.750	20.0	95.0	115.0	18.0	14
Quincy Acy	24098	2017.0	PF	26.0	DAL	6.0	0.0	48.0	-1.4	0.355	...	0.667	2.0	6.0	8.0	0.0	0
Quincy Acy	24099	2017.0	PF	26.0	BRK	32.0	1.0	510.0	13.1	0.587	...	0.754	18.0	89.0	107.0	18.0	14
Steven Adams	24100	2017.0	C	23.0	OKC	80.0	80.0	2389.0	16.5	0.589	...	0.611	282.0	333.0	615.0	86.0	88
Arron Afflalo	24101	2017.0	SG	31.0	SAC	61.0	45.0	1580.0	9.0	0.559	...	0.892	9.0	116.0	125.0	78.0	20
Alexis Ajinca	24102	2017.0	C	28.0	NOP	39.0	15.0	584.0	12.9	0.529	...	0.725	46.0	131.0	177.0	12.0	20
Cole Aldrich	24103	2017.0	C	28.0	MIN	62.0	0.0	531.0	12.7	0.549	...	0.682	51.0	107.0	158.0	25.0	29

Inversão linhas x colunas

Podemos inverter as linhas e colunas de um Dataframe solicitando a matriz Transposta

dados.T											
Player	Curly Armstrong	Cliff Barker	Leo Barnhorst	Ed Bartels	Ed Bartels	Ed Bartels	Ralph Beard	Gene Berce	Charlie Black	Charlie Black	...
Unnamed: 0	0	1	2	3	4	5	6	7	8	9	...
Year	1950	1950	1950	1950	1950	1950	1950	1950	1950	1950	...
Pos	G-F	SG	SF	F	F	F	G	G-F	F-C	F-C	...
Age	31	29	25	24	24	24	22	23	28	28	...
Tm	FTW	INO	CHS	TOT	DNN	NYK	INO	TRI	TOT	FTW	...
G	63	49	67	15	13	2	60	3	65	36	...
GS	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
MP	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
PER	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
TS%	0.368	0.435	0.394	0.312	0.308	0.376	0.422	0.275	0.346	0.362	...

Tratamento de Valores NaN

Substituir todos os NaN por Zero:

```
b = dados.GS
dados.GS.fillna(0)
```

Player	
Curly Armstrong	0.0
Cliff Barker	0.0
Leo Barnhorst	0.0
Ed Bartels	0.0
Ed Bartels	0.0
Ed Bartels	0.0
Ralph Beard	0.0
Gene Berce	0.0
Charlie Black	0.0
Charlie Black	0.0
Charlie Black	0.0
Nelson Bobb	0.0
Jake Bornheimer	0.0
Vince Boryla	0.0

Tratamento de Valores NaN

Mostrar todos os valores NaN

```
dados.isnull()
```

	Unnamed: 0	Year	Pos	Age	Tm	G	GS	MP	PER	TS%	...	FT%
Player												
Curly Armstrong	False	False	False	False	False	False	True	True	True	False	...	False
Cliff Barker	False	False	False	False	False	False	True	True	True	False	...	False
Leo Barnhorst	False	False	False	False	False	False	True	True	True	False	...	False
Ed Bartels	False	False	False	False	False	False	True	True	True	False	...	False
Ed Bartels	False	False	False	False	False	False	True	True	True	False	...	False

Tratamento de Valores NaN

Apagar valores NaN

```
dados.dropna()
```

```
      Unnamed: 0  Year  Pos  Age
```

Player

0 rows × 52 columns

Referências

Livro: Python for Data Analysis

Livro: Python Data Science Handbook

<https://www.datasciencecentral.com/profiles/blogs/top-20-python-libraries-for-data-science-in-2018>

<https://becode.com.br/porque-aprender-python/>

<http://mindbending.org/pt/a-historia-do-python>

<https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages>

<https://blog.liveedu.tv/top-3-most-popular-programming-languages-2018/>