Book - Data Science

13 Nov 2017 –



Sumário

[Nanodegree Fundamentos de Data Science I 3](#_Toc501637608)

[Introdução 3](#_Toc501637609)

[Rodando códigos Python no Atom 3](#_Toc501637610)

[Resumo: Rodando códigos na linha de comando 3](#_Toc501637611)

[Resumo dos comandos 3](#_Toc501637612)

[Tipos Básicos de Variáveis 4](#_Toc501637613)

[Números 4](#_Toc501637614)

[Inteiros 4](#_Toc501637615)

[Floats 4](#_Toc501637616)

[Procedure 4](#_Toc501637617)

[Função Index 5](#_Toc501637618)

[Dicionário – immutable – Mutable – String, List, Dicionary 6](#_Toc501637619)

[dictionary implementation 6](#_Toc501637620)

[Tuplas 7](#_Toc501637621)

[Por que tuplas? 7](#_Toc501637622)

[Descompactando uma tupla 7](#_Toc501637623)

[Imutabilidade de tupla 8](#_Toc501637624)

[Retorno de tuplas 8](#_Toc501637625)

[A biblioteca padrão 8](#_Toc501637626)

[Importação de módulos 8](#_Toc501637627)

[Outras formas de importar e nomear 9](#_Toc501637628)

[Para importar uma função ou classe individual de um módulo use 9](#_Toc501637629)

[Para importar um módulo e atribuir um nome diferente (geralmente mais curto) 9](#_Toc501637630)

[Importar um item individual de um módulo e atribuir um nome diferente 9](#_Toc501637631)

[Nossos módulos favoritos 9](#_Toc501637632)

[Instalando pacotes com pip 10](#_Toc501637633)

[Pacotes úteis de terceiros 11](#_Toc501637634)

[requirements.txt 11](#_Toc501637635)

[Usando arquivos 12](#_Toc501637636)

[Abertura e leitura de arquivos 12](#_Toc501637637)

[Escrevendo em um arquivo 13](#_Toc501637638)

[with 13](#_Toc501637639)

[Exception 15](#_Toc501637640)

[Instalando o Anaconda 15](#_Toc501637641)

[No Windows 15](#_Toc501637642)

[Data Analysis Processes 16](#_Toc501637643)

[Lendo arquivos CSV 17](#_Toc501637644)

[Cabeçalhos 17](#_Toc501637645)

[Índice 18](#_Toc501637646)

[Gravando arquivos CSV 18](#_Toc501637647)

[Avaliando e desenvolvendo intuição 18](#_Toc501637648)

[Indexando e selecionando dados no Pandas 19](#_Toc501637649)

[Prática de limpeza 20](#_Toc501637650)

[Renomeando colunas 21](#_Toc501637651)

[Renomear mais eficiente 21](#_Toc501637652)

[Plotting com o Pandas 21](#_Toc501637653)

[Explorando dados com visualização 21](#_Toc501637654)

[NumPy - Appending 23](#_Toc501637655)

[Adicionando colunas 24](#_Toc501637656)

[Anexando dados 24](#_Toc501637657)

[Crie colunas de cor 24](#_Toc501637658)

[The Python Standard Library 24](#_Toc501637659)

[9.2. math — Mathematical functions 24](#_Toc501637660)

# Nanodegree Fundamentos de Data Science I

# Introdução

Habilidades necessárias CECOU SQL, ou outras linguagens como Hadoop ou MapReduce, estudar pacotes estatísticos para analisar e modelar dados como como R ou Python.

## Rodando códigos Python no Atom

Para rodar códigos em Python e ver o resultado diretamente no aplicativo Atom, siga os passos a seguir:

Passo 1:

Certifique-se de que seu código está salvo com uma extensão de arquivo .py(caso contrário, a construção não funcionará).

Passo 2:

Para rodar (ou “construir”) seu código no Atom, você precisará instalar um pacote. Usaremos o pacote script e o instalaremos da seguinte forma:

Navegue para barra de menu, no topo da janela (atenção: se a barra de menu estiver oculta, utiliza a tecla ALT em seu teclado para alterar sua visibilidade). Na barra de menu, seleciona File → Settings. Você deverá ver a aba de configurações surgir, contendo algumas opções na barra lateral esquerda. Clique na opção marcada como Install.

No campo de buscas, digite script e pressione ENTER em seu teclado.

Localize o pacote chamado script. Perceba que o nome do pacote também pode conter sua versão (ex: 3.13.0) e uma pequena descrição (ex: “Rode código no Atom!”). Clique no botão Install desse pacote.

Quando o pacote script estiver instalado, você poderá fechar a aba de configurações e retornar para a aba contendo o arquivo Python. Para rodar (ou “construir”) o código, navegue para a barra de menu e selecione Packages → Script → Run Script. Você também pode usar o atalho de teclado: Control + Shift+ b. Como resultado, você verá um painel na parte de baixo da janela do Atom (conforme mostrado abaixo). Para fechar esse painel, simplesmente aperte a tecla ESC em seu teclado.

# Resumo: Rodando códigos na linha de comando

## Resumo dos comandos

No tutorial anterior, sobre linha de comando, você aprendeu a executar os seguintes comandos:

* ls, que **l**ista o conteúdo de um diretório,
* cd, que modifica (**c**hanges) o **d**iretório em uso,
* pwd, que imprime (**p**rints) o diretório em uso (**w**orking **d**irectory),
* open/start, que abre (**o**pens) um arquivo (no Mac ou Windows, respectivamente),
* touch, que cria um novo arquivo com uma extenão específica ou tipo de arquivo,
* mkdir, que cria (**m**a**k**es) um novo **dir**etório,
* rm, que **r**e**m**ove permanentemente um arquivo,
* rmdir, que **r**e**m**ove permanentemente um **dir**etótio vazio,
* rm -r, que **r**e**m**ove um diretório e seu conteúdo (sem confirmação),
* rm -ri, que permanentemente **r**e**m**ove um diretório e seu conteúdo (com confirmação).

# Tipos Básicos de Variáveis

## Números

### Inteiros

Um número inteiro é um número inteiro. Em teoria, todo o número inteiro de menos infinito para mais infinito. Na prática, o alcance é limitado pelo computador, mas com o Python o tamanho máximo é limitado apenas pela quantidade de memória que seu computador possui!

### Floats

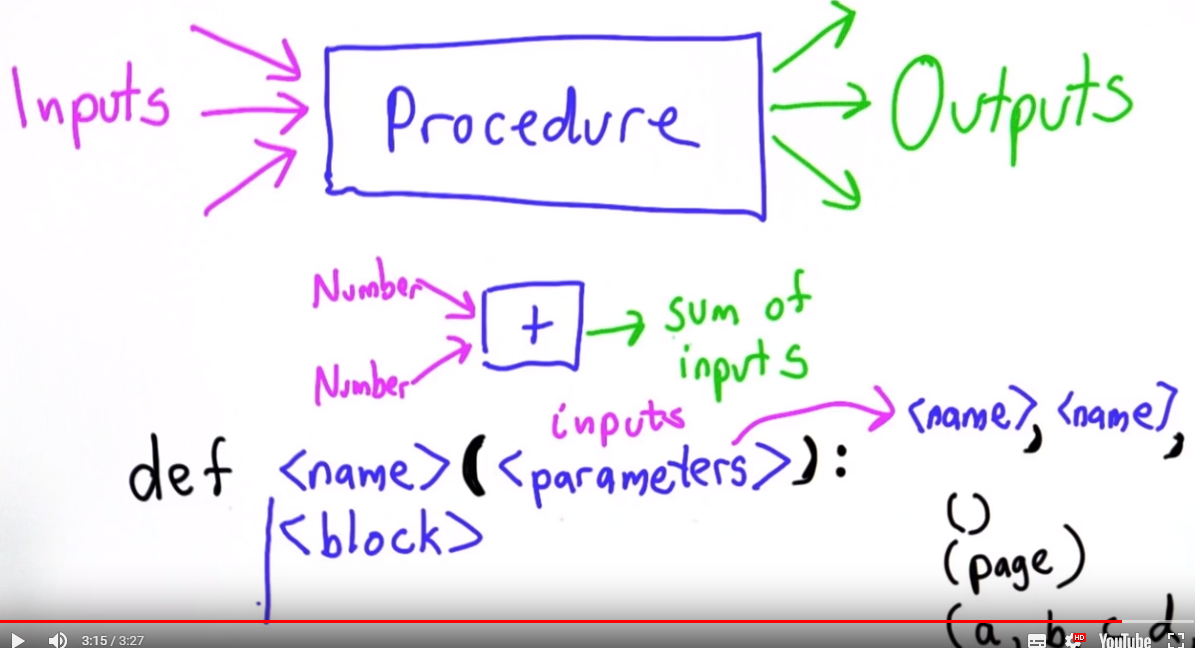
Os computadores são muito bons em lidar com números inteiros. É por isso que o 'inteiro' é um tipo de dados básico. Um "número de ponto flutuante" é qualquer número que tenha dígitos após o ponto decimal. Por exemplo, 6.3 ou -1.8.

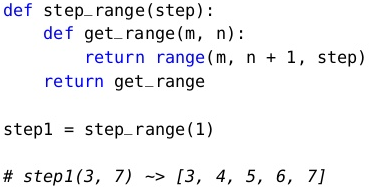
Você também pode armazenar seus números inteiros como números de ponto flutuante, basta incluir um ponto decimal quando você o escreve: 1.0 ou -6.0. Porque lidar com números de ponto flutuante é muito mais complicado para os computadores, eles são muito mais lentos para trabalhar. Se você sabe que seus números serão inteiros, use o tipo inteiro.

Se você precisa de muitas matemáticas complicadas com números de ponto flutuante, então você pode estar interessado em extensões como o numpy ou o projeto Multipccision Python Geral. Essas extensões fornecem maneiras muito rápidas de fazer operações matemáticas. Para uso normal, os recursos criados para o Python serão suficientes.

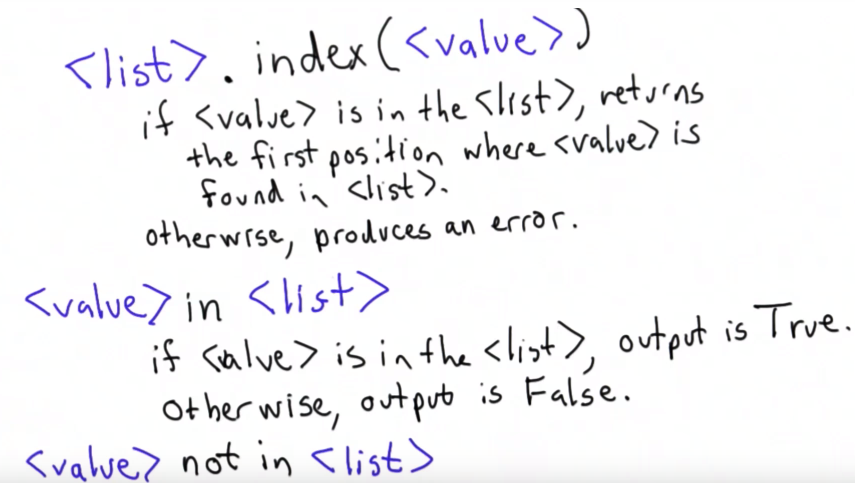
# Procedure

Modelo:





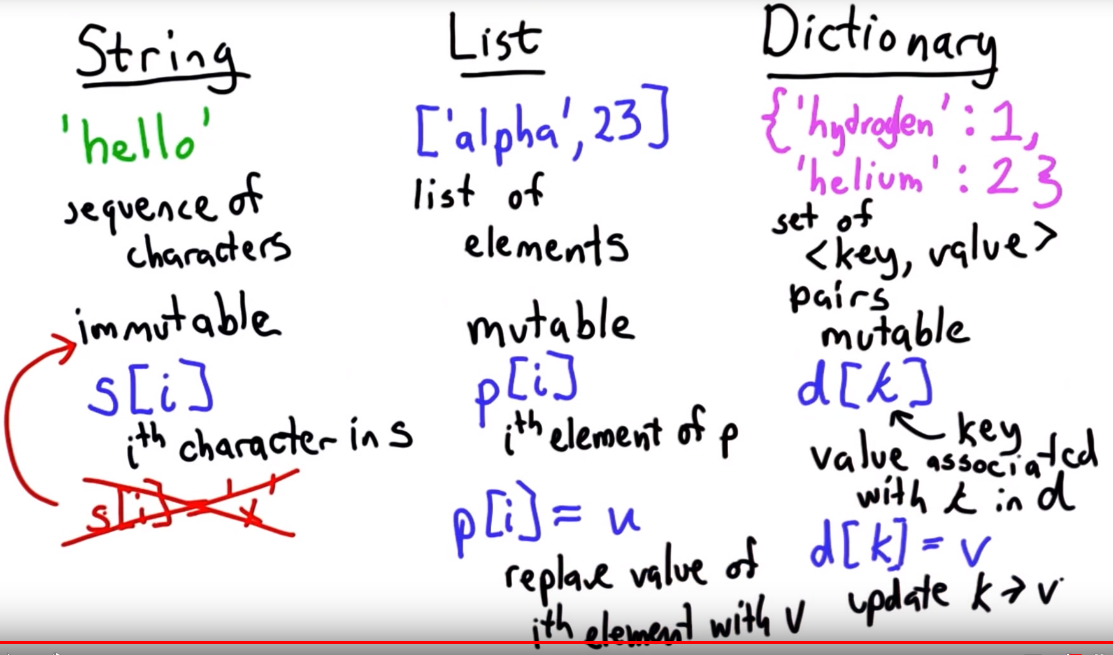
# Função Index

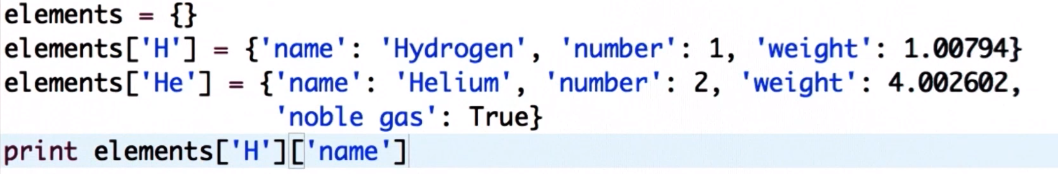


## Dicionário – immutable – Mutable – String, List, Dicionary

### [dictionary implementation](http://www.laurentluce.com/posts/python-dictionary-implementation/)

<http://www.laurentluce.com/posts/python-dictionary-implementation/>





# Tuplas

O Python fornece outro utilitário embutido útil: as tuplas. Elas são usadas para armazenar peças de informação relacionadas. Considere este exemplo envolvendo latitude e longitude:

Referencias:

<https://pt.stackoverflow.com/questions/192727/quando-usar-listas-e-quando-usar-tuplas>

>>> AngkorWat = (13.4125, 103.866667)

>>> print(type(AngkorWat))

<class 'tuple'>

>>> print("Angkor Wat is at latitude: {}".format(AngkorWat[0]))

Angkor Wat is at latitude: 13.4125

>>> print("Angkor Wat is at longitude: {}".format(AngkorWat[1]))

Angkor Wat is at longitude: 103.866667

As tuplas são semelhantes às listas em que armazenam uma coleção ordenada de objetos que podem ser acessados ​​por seus índices (por exemplo AngkorWat[0] e AngkorWat[1]). Ao contrário das listas, tuplas são imutáveis. Você não pode adicionar e remover itens de tuplas ou colocá-los no lugar.

## Por que tuplas?

Por que nós temos tuplas, se elas são semelhantes às listas, mas com menos recursos? Elas são úteis quando você tem dois ou mais valores que estão tão intimamente relacionados, que sempre serão usados ​​juntos, como coordenadas de latitude e longitude.

As tuplas podem ser usadas para atribuir múltiplas variáveis ​​de forma compacta:

>>> dimensions = 52, 40, 100

>>> length, width, height = dimensions

>>> print("The dimensions are {}x{}x{}".format(length, width, height))

The dimensions are 52x40x100

Observe que os valores atribuídos à tupla dimensions não estão rodeados de parênteses, como eram os exemplos anteriores. Os parênteses são opcionais quando se fazem tuplas, e programadores frequentemente os omitem se parênteses não esclarecem o código.

## Descompactando uma tupla

Na segunda linha, três variáveis ​​são atribuídas a partir do conteúdo da tupla dimensions. Isso é chamado de descompactamento de tupla. Você pode usar tuplas descompactadas ou atribuir as informações de uma tupla a diversas variáveis ​​sem ter que acessá-las uma a uma e criar várias instruções de atribuição.

Neste exemplo, se não precisarmos usar dimensions diretamente, poderíamos encurtar essas duas linhas de código a uma única linha que atribui três variáveis ​​de uma só vez!

length, width, height = 52, 40, 100

## Imutabilidade de tupla

Há também um lugar onde a imutabilidade da tupla é uma vantagem. Ao contrário das listas, as tuplas podem ser armazenadas em conjuntos ou usadas como as chaves de um dicionário. Uma vez que essas duas estruturas de dados requerem chaves imutáveis, as listas não são uma opção (se você está curioso para saber por que conjuntos e dicionários exigem chaves imutáveis, pode aprender isso no [**CS101 - aula 5**](https://classroom.udacity.com/courses/cs101/lessons/48682650/concepts/486995980923). Nesta aula, você constrói um dicionário do zero para aprender como eles funcionam). No exemplo abaixo, criamos um dicionário, world\_heritage\_locations que tem tuplas da forma (latitude, longitude) com chaves e valores que indicam o nome de locais correspondentes.

world\_heritage\_locations = {(13.4125, 103.866667): "Angkor Wat",

(25.73333, 32.6): "Ancient Thebes",

(30.330556, 35.4433330): "Petra",

(-13.116667, -72.583333): "Machu Picchu"}

## Retorno de tuplas

Um uso comum para tuplas é retornar vários valores de uma função:

**def** **first\_and\_last**(sequence):

"""Retorna o primeiro e o último elemento de uma sequência"""

**return** sequence[0], sequence[-1]

# A biblioteca padrão

Pense na biblioteca padrão do Python como um grande conjunto de ferramentas de programação que você pode usar para ajudar na programação em Python. Ela fornece novos tipos de objetos e funções para uma série de tarefas comuns e especializadas, de redes a estatística matemática. Outras pessoas já escreveram esse código e o colocaram em "módulos" úteis para você acessar e usar em seu próprio código. Usando módulos da biblioteca padrão do Python para acessar e usar facilmente o código existente, você tem muito poder de programação!

Nesta seção, faremos uma rápida visita à biblioteca padrão, ensinaremos você a usá-la e compartilharemos alguns de nossos módulos favoritos.

## Importação de módulos

A biblioteca padrão do Python é organizada em partes chamadas modules. Muitos módulos são simplesmente arquivos do Python, como os scripts de Python que você já usou e escreveu. Para poder usar o código contido em um módulo, é necessário importá-lo, seja no interpretador interativo ou em um script próprio de Python.

A sintaxe para importar um módulo é simplesmente import package\_name.

>>> **import** math

## Outras formas de importar e nomear

Até agora, você importou módulos com import module\_name, que torna todas as classes de objetos e funções desse módulo disponíveis via notação de ponto.

Existem algumas outras variantes de importação que são úteis em outras situações.

### Para importar uma função ou classe individual de um módulo use

>>> **from** module\_name **import** object\_name

por exemplo:

>>> **from** collections **import** defaultdict

>>> **from** collections **import** defaultdict, namedtuple

### Para importar um módulo e atribuir um nome diferente (geralmente mais curto)

>>> **import** module\_name **as** different\_name

por exemplo

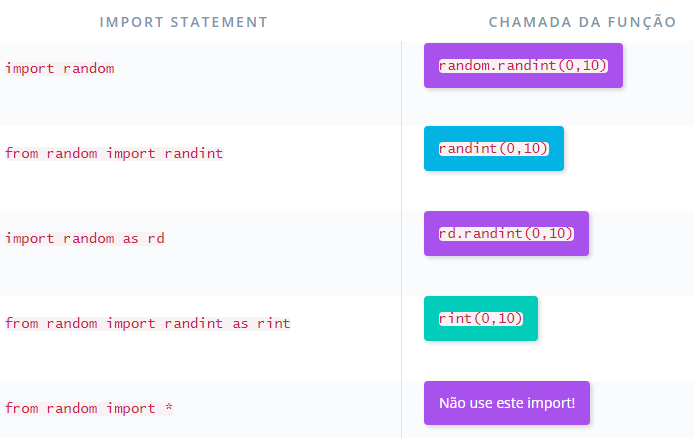
>>> **import** multiprocessing **as** mp

### Importar um item individual de um módulo e atribuir um nome diferente

**from** module\_name **import** object\_name **as** different\_name

Por exemplo:

**from** csv **import** reader **as** csvreader



## Nossos módulos favoritos

A biblioteca padrão do Python tem muitos módulos! Para ajudar você a se familiarizar com o que está disponível, aqui está uma seleção dos nossos módulos de biblioteca padrão do Python e por que os usamos!

* [**csv**](https://docs.python.org/3/library/csv.html): muito conveniente para ler e escrever arquivos csv
* [**collections**](https://docs.python.org/3/library/collections.html): extensões úteis dos tipos de dados habituais, incluindo OrderedDict, defaultdict e namedtuple
* [**random**](https://docs.python.org/3/library/random.html): gera números pseudo-aleatórios, embaralha sequências aleatoriamente e escolhe itens aleatórios
* [**string**](https://docs.python.org/3/library/string.html): mais funções em strings. Este módulo também contém coleções úteis de letras como string.digits(uma string contendo todos os caracteres com dígitos válidos).
* [**re**](https://docs.python.org/3/library/re.html): padronização em strings através de expressões regulares
* [**math**](https://docs.python.org/3/library/math.html): algumas funções matemáticas padrão
* [**os**](https://docs.python.org/3/library/os.html): interagindo com sistemas operacionais
* [**os.path**](https://docs.python.org/3/library/os.path.html): submódulo de os para manipular nomes de caminho
* [**sys**](https://docs.python.org/3/library/sys.html): trabalhar diretamente com o interpretador Python
* [**json**](https://docs.python.org/3/library/json.html): bom para ler e escrever arquivos json (bom para o trabalho na web)

# Instalando pacotes com pip

Vamos usar o pip para instalar a biblioteca pytz. Para garantir que estamos usando o pip associado à instalação do Python 2, usaremos o comando pip2. A pytz é uma biblioteca para trabalhar com fusos horários, que é [**uma tarefa extremamente complicada**](http://infiniteundo.com/post/25326999628/falsehoods-programmers-believe-about-time).

Podemos instalar pytz a partir da nossa linha de comando, com pip3:

$ pip3 install pytz

Esse comando irá baixar e instalar pytz, para que esteja disponível para importação em nossos programas. Uma vez instalado, podemos importar pacotes de terceiros usando a mesma sintaxe que usamos para importar da biblioteca padrão. Neste exemplo, importarei pytz e também datetime da biblioteca padrão. É prática padrão colocar as declarações de importação para bibliotecas de terceiros após as importações da biblioteca padrão.

**from** datetime **import** datetime

**import** pytz

utc = pytz.utc *# utc significa "tempo universal coordenado"*

ist = pytz.timezone('Asia/Kolkata') *# IST é hora padrão da Índia*

now = datetime.datetime.now(tz=utc) *# é a hora atual em UTC*

ist\_now = now.astimezone(ist) *# é a hora atual em IST.*

Este exemplo armazena a hora atual, expressa em termos de [**tempo universal coordenado**](https://en.wikipedia.org/wiki/Coordinated_Universal_Time), na variável now. Em seguida, ele traduz esse tempo em horário padrão indiano e o armazena na variável ist\_now.

## Pacotes úteis de terceiros

Ser capaz de instalar e importar bibliotecas de terceiros é útil, mas, para ser um programador eficaz, você também precisa saber quais bibliotecas estão disponíveis para uso. As pessoas geralmente aprendem sobre novas bibliotecas úteis com outras pessoas - por uma recomendação on-line ou de um colega. Se você é novo na programação em Python, pode não ter muitos colegas na área. Então, para começar, aqui está uma lista de pacotes que são populares entre os engenheiros da Udacity.

* [**IPython**](https://ipython.org/) - Um melhor interpretador interativo de Python.
* [**requests**](http://docs.python-requests.org/) - Fornece métodos fáceis de usar para fazer solicitações na web.
* [**Flask**](http://flask.pocoo.org/) - Uma estrutura leve para fazer aplicações web e APIs.
* [**Django**](https://www.djangoproject.com/) - Uma estrutura com mais recursos para fazer aplicações web. O Django é particularmente bom para projetar, usar com conteúdo pesado e aplicações web.
* [**Beautiful Soup**](https://www.crummy.com/software/BeautifulSoup/) - Usado para analisar HTML e extrair informações dele. Ótimo para web scraping.
* [**pytest**](http://doc.pytest.org/) - Estende Python construído em asserções e módulo de teste de unidade.
* [**PyYAML**](http://pyyaml.org/wiki/PyYAML) - Para ler e escrever arquivos [**YAML**](https://en.wikipedia.org/wiki/YAML).
* [**NumPy**](http://www.numpy.org/) - O pacote fundamental para computação científica com Python. Contém, entre outras coisas, um poderoso objeto de matriz N-dimensional e capacidades úteis de álgebra linear.
* [**pandas**](http://pandas.pydata.org/) - Uma biblioteca contendo alto desempenho, estruturas de dados e ferramentas de análise de dados. Em particular, o Pandas fornece dataframes!
* [**matplotlib**](http://matplotlib.org/) - Uma biblioteca 2D de plotagem que produz qualidade de publicação de figuras em uma variedade de formatos impressos e ambientes interativos.
* [**ggplot**](http://ggplot.yhathq.com/) - Outra biblioteca de plotagem 2D, baseada na biblioteca ggplot2 de R.
* [**Pillow**](https://python-pillow.org/) - Uma biblioteca de imagens do Python que adiciona recursos de processamento de imagem ao seu intérprete Python.
* [**pyglet**](http://www.pyglet.org/) - Um framework de aplicação multiplataforma, destinado ao desenvolvimento de jogos.
* [**Pygame**](http://www.pygame.org/) - Um conjunto de módulos Python projetado para escrever jogos.
* [**pytz**](http://pytz.sourceforge.net/) - Definições de fusos-horários mundial para Python.

## requirements.txt

Os programas Python maiores podem depender de dezenas de pacotes de terceiros. Para tornar mais fácil compartilhá-los, os programadores geralmente listam as dependências de um projeto em um arquivo chamado requirements.txt. Este é um exemplo de um arquivo requirements.txt:

beautifulsoup4==4.5.1

bs4==0.0.1

pytz==2016.7

requests==2.11.1

Cada linha do arquivo inclui o nome de um pacote e o número de sua versão. O número da versão é tecnicamente opcional, mas, geralmente, deve ser incluído. As bibliotecas podem mudar sutilmente (ou dramaticamente!) entre as versões, por isso, é importante usar as mesmas versões de biblioteca que o autor do programa tinha quando escreveu o programa.

Você pode usar pip para instalar todas as dependências de um projeto de uma vez, com este comando:

$ pip3 install -r requirements.txt

# Usando arquivos

Para que um programa seja realmente útil, ele precisa interagir com dados do mundo real. Imagens, páginas da web, bancos de dados são todos exemplos de arquivos e, rotineiramente, criamos, movemos, manipulamos e lemos esses arquivos em nossa vida digital diária.

Todos os dados que usamos até agora foram definidos dentro do interpretador Python ou de um script Python. Em seguida, aumentaremos massivamente a variedade do que podemos alcançar com a programação em Python, incluindo como abrir e ler arquivos em Python. Isso nos permitirá interagir e processar grandes quantidades de informação de qualquer fonte.

Todos os tipos de arquivo têm uma estrutura semelhante em um computador - são sequências de caracteres que codificam algumas informações. O formato de arquivo específico (frequentemente indicado pela extensão do nome do arquivo, tal como .txt or .mp3) indicará como esses personagens são organizados. Os caracteres no arquivo são interpretados pelos vários programas que usamos para trabalhar com eles - por exemplo, um programa de edição de imagem interpretará as informações de um arquivo de fotografia digital e exibirá a imagem. Se, em seguida, editarmos a imagem no programa, estamos usando o programa para fazer alterações nos caracteres do arquivo.

Em Python, podemos ler esses caracteres de arquivo diretamente. A experiência parecerá bastante diferente de abrir um arquivo em um aplicativo de desktop. Abrir arquivos em Python nos dá uma interface programática comum para todos os tipos de arquivo, sem a necessidade de uma interface gráfica de usuário - o que significa que podemos automatizar tarefas envolvendo arquivos com programas em Python.

Aqui está um exemplo de código para ler informações de um arquivo em Python.

## Abertura e leitura de arquivos

A fim de ler um arquivo, devemos primeiro abri-lo, e podemos fazê-lo com a função open. Devemos dar-lhe um caminho onde o arquivo reside e também podemos especificar vários parâmetros opcionais. Podemos, então, atribuir o valor retornado da função para uma variável. Por exemplo:

f = open('/my\_path/my\_file.txt','r')

A função open retornará um objeto de arquivo - um objeto Python pelo qual o Python interagirá com o próprio arquivo.

Neste exemplo, o segundo parâmetro r especifica o modo no qual abrimos o arquivo - no nosso caso, abrimos em modo somente leitura (porque só queremos ler o arquivo, não alterar seu conteúdo). Não precisamos especificar esse parâmetro, porque, por padrão (se não especificar o modo), o open abrirá os arquivos no modo de leitura.

Em seguida, para acessar o conteúdo do arquivo, podemos usar read. Af.read() cria um objeto string contendo o texto no arquivo. Essa string é atribuída à variável nomeada file\_data.:

file\_data = f.read()

Assim que terminarmos o arquivo f, devemos fechá-lo. Isso liberará todos os recursos do sistema ocupados pelo arquivo:

f.close()

É importante lembrarmos de sempre fechar os arquivos que abrimos quando não precisamos mais deles. Se abrimos muitos arquivos sem fechá-los, podemos ficar sem manipuladores e não seremos capazes de abrir novos arquivos (exatamente quantos arquivos você pode abrir para ainda conseguir manipulá-los dependerá de seu sistema operacional).

Para se convencer, você pode tentar executar o seguinte em seu interpretador python:

>>> files = []

>>> **for** i **in** range(100000):

... files.append(open('somefile.txt'))

Tente editar o número em range, no laço for. Em algum ponto, para um número suficientemente grande, você receberá um erro.

Traceback (most recent call last):

File "<stdin>", line 2, **in** <module>

OSError: [Errno 24] Too many open files: 'somefile.txt'

>>> i

7164

Podemos ver que em 7.164 arquivos abertos o sistema não havia mais recursos disponíveis para abrir novos arquivos. Para evitar isso, é sempre uma boa ideia fechar todos os arquivos de que você não precisa mais.

## Escrevendo em um arquivo

Além de ler um arquivo, você também pode gravar algo em um arquivo, se quiser alterar seu conteúdo. Para fazer isso, você deve abrir o arquivo no modo de gravação:

f = open('/my\_path/my\_file.txt','w')

Cuidado: quando um arquivo é aberto no modo de gravação, qualquer coisa contida anteriormente será excluída. Caso esteja interessado em adicionar algo a um arquivo existente (sem excluir seu conteúdo), você deve usar append em vez de write e abri-lo no modo de anexação (usando a em vez de w). Você pode consultar a [**documentação do Python**](https://docs.python.org/3/library/functions.html#open) para obter mais informações sobre os diferentes modos nos quais um arquivo pode ser aberto.

Se o arquivo não existir, o Python o criará para você.

Agora, podemos escrever no arquivo:

f.write("Hello World!")

E, depois que terminarmos, vamos fechá-lo, como bons cidadãos.

f.close()

## with

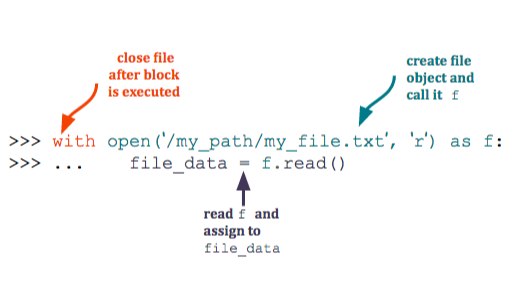
O Python permite abrir um arquivo, fazer operações nele e fechá-lo automaticamente depois, usando with.

>>> **with** open('/my\_path/my\_file.txt','r') **as** f:

>>> file\_data = f.read()

No exemplo acima, abrimos um arquivo, executamos as operações no bloco abaixo da declaração with (nesse caso, ler no arquivo) e depois o Python o fecha para nós. Não há necessidade de usar f.close().

Isso é explicado em mais detalhes abaixo.



**with** open('/my\_path/my\_file.txt','r') **as** f:

file\_data = f.read()

Neste código, a função open usa como entrada um caminho para um arquivo ('/my\_path/my\_file.txt') no sistema de arquivos e cria um objeto de arquivo. Como já vimos, o 'r' significa que esse objeto pode ser usado somente para o modo de **leitura**. O r' é realmente o modo padrão, portanto, não é estritamente necessário ao chamar open.

O código as f atribui o objeto de arquivo criado pela chamada para open para o nome da variável f - é semelhante a dizer f = open('/my\_path/my\_file.txt','r').

Na chamada f.read(), o objeto de arquivo f lê (todo) o conteúdo do arquivo subjacente /my\_path/my\_file.txt, e então f.read() cria um objeto string contendo esse texto. Essa string é atribuída à variável nomeada file\_data.

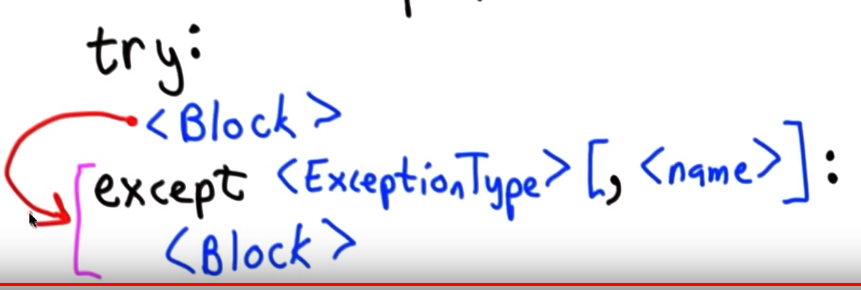
Poderia ser fácil esquecer de fechar o arquivo quando você está codificando, mas o Python fornece uma sintaxe especial que o fecha automaticamente: a palavra-chave éwith.

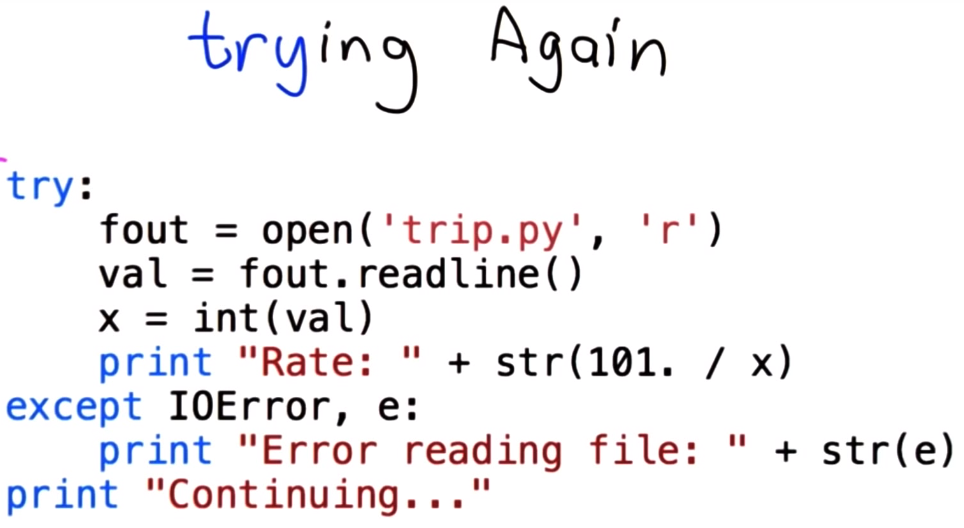
Depois de uma instrução with open(filename) as f: (não se esqueça dos dois pontos no final), escreva um bloco recuado para usar esse objeto de arquivo aberto f. Uma vez executado o bloco de recuo, o arquivo será fechado automaticamente. Esse é outro tipo de escopo - você só pode acessar os dados no arquivo via f dentro desse bloco recuado. Uma vez fechado o arquivo, você será incapaz de interagir com ele.

A documentação oficial para open e objetos de arquivo pode ser encontrada [**aqui**](https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files) e contém mais exemplos e explicações.

Após o código de exemplo ter sido executado, a string file\_data contém todo o arquivo em uma única sequência. Você pode usar todos os métodos de string usuais nessa string e processar seu conteúdo.

# Exception





# Instalando o Anaconda

O Anaconda está disponível para Windows, Mac OS X e Linux. É possível encontrar os instaladores e as instruções de instalação no site <https://www.continuum.io/downloads>.

Depois da instalação, você estará automaticamente no ambiente conda padrão com todos os pacotes instalados, como mostrado abaixo. É possível checar sua instalação digitando conda list no terminal.

## No Windows

Diversas aplicações são instaladas junto ao Anaconda:

**Anaconda Navigator**, uma interface para gerenciar seus pacotes e ambientes

**Anaconda Prompt**, um terminal para usar a interface de linha de comando para gerenciar seus ambientes e pacotes

**Spyder**, uma interface equipada para o desenvolvimento científico

Para evitar problemas futuros, é melhor atualizar todos os pacotes do ambiente padrão. Abra o programa **Anaconda Prompt**, digite os seguintes comandos:

**conda** upgrade conda

**conda** upgrade --all

e confirme quando for pedido. Os pacotes que vêm com a instalação inicial tendem a não ser a versão mais atual. Ao fazer essa atualização agora, você evita erros futuros causados por pacotes antigos.

**Observação:** na etapa anterior, rodar conda upgrade conda não deveria ser necessário, pois o comando seguinte contém --all, que deveria incluir o pacote conda, mas alguns usuários reportaram erros ao não fazer isso.

Para o restante desta aula, será necessário digitar diversos comandos no terminal. Recomendo fortemente que comece a trabalhar com o Anaconda desta maneira e só depois passe a usar uma interface gráfica, caso você queira.

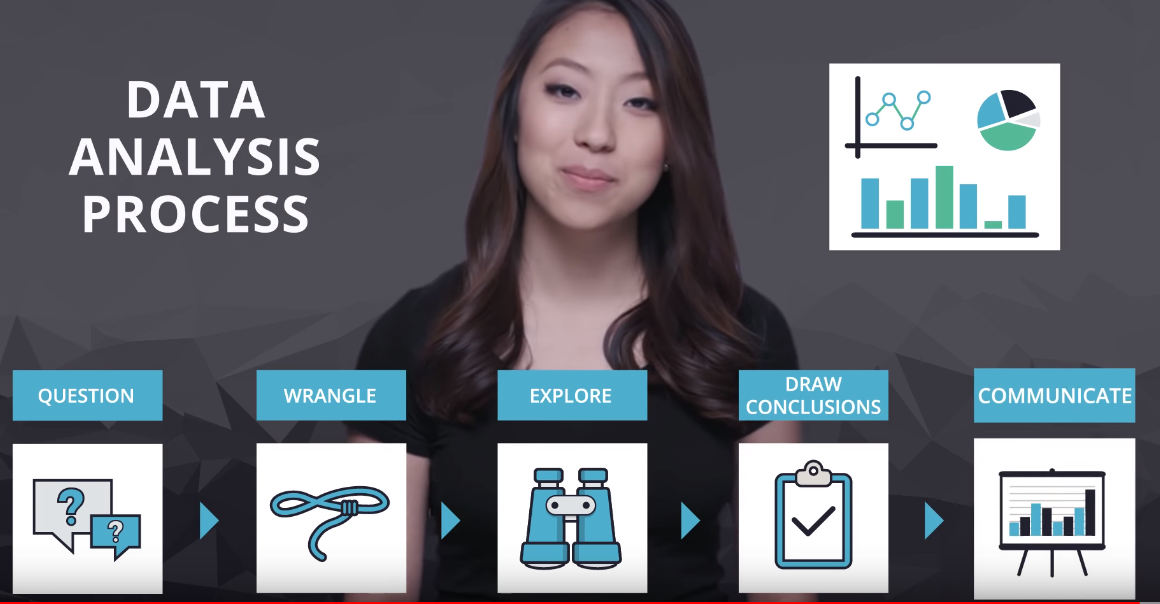
Erros

Caso você esteja vendo a mensagem "conda command not found" e estiver usando o ZShell, é necessário fazer o seguinte:

* Adicione export PATH="/Users/username/anaconda/bin:$PATH" ao arquivo ".zsh\_config".

# Data Analysis Processes

5 Steps.



# Lendo arquivos CSV

Vamos praticar a leitura de arquivos csv com este conjunto de dados fictício com notas de alunos. Como você já viu algumas vezes, read\_csv() é usado para carregar dados de arquivos csv em um dataframe do Pandas. Temos apenas que especificar o caminho do arquivo que contém nossos dados. Eu armazenei student\_scores.csv no mesmo diretório que este notebook Jupyter, então temos apenas que fornecer o nome do arquivo.

**import** pandas **as** pd

​

df = pd.read\_csv('student\_scores.csv')

head() é uma função útil que você pode chamar no seu dataframe para exibir as primeiras linhas. Vamos usá-la para ver como os dados estão organizados.

## Cabeçalhos

Outra coisa que você pode fazer com read\_csv é especificar que linha do arquivo é o cabeçalho, que especifica os rótulos de cada coluna. Normalmente se usa a primeira linha para isso, mas pode ser que se queira escolher uma linha mais abaixo caso haja informações adicionais no topo do arquivo. Podemos fazer isso assim.

df = pd.read\_csv('student\_scores.csv', header=2)

df.head()

Aqui, a linha 2 foi usada como o cabeçalho e tudo que estava nas linhas acima foi ignorado. Como padrão, read\_csv usa header=0, que usa a primeira linha como cabeçalho das colunas.

Se rótulos de colunas não foram incluídos no seu arquivo, você pode usar header=None para previnir que sua primeira linhas de dados seja interpretada de forma incorreta como rótulos de colunas.

df = pd.read\_csv('student\_scores.csv', header=**None**)

df.head()

Você também pode especificar seus próprios rótulos de colunas assim.

labels = ['id', 'name', 'attendance', 'hw', 'test1', 'project1', 'test2', 'project2', 'final']

df = pd.read\_csv('student\_scores.csv', names=labels)

df.head()

Se você quiser dizer para o pandas que havia um cabeçalho que você está substituindo, você pode especificar a linha do cabeçalho substituído assim.

labels = ['id', 'name', 'attendance', 'hw', 'test1', 'project1', 'test2', 'project2', 'final']

df = pd.read\_csv('student\_scores.csv', header=0, names=labels)

df.head()

## Índice

Ao invés de usar o índice padrão (inteiros incrementados de 1 a partir do 0), você pode especificar uma ou mais colunas para serem o índice do seu dataframe.

df = pd.read\_csv('student\_scores.csv', index\_col='Name')

df.head()

df = pd.read\_csv('student\_scores.csv', index\_col=['Name', 'ID'])

df.head()

Você pode fazer muito mais que isso simplesmente usando esta função, como separar datas, preencher valores nulos, pular linhas, etc. Muitas destas coisas podem ser feitas com diferentes passos após a chamada da função read\_csv(). Vamos modificar nossos dados de outras formas, mas você sempre pode consultar como fazer alguns destes passos com esta função [aqui](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html).

## Gravando arquivos CSV

Legal! Agora vamos salvar seu segundo dataframe com dados da usina elétrica em um arquivo csv para a próxima seção.

df\_powerplant.to\_csv('powerplant\_data\_edited.csv')

Vamor ver se isso funcionou da forma que gostaríamos.

df = pd.read\_csv('powerplant\_data\_edited.csv')

df.head()

O que é esse Unnamed:0? to\_csv() armazenará nosso índice a não ser que digamos explicitamente que ele não deve fazer isso. Para fazê-lo ignorar o índice, nós temos que fornecer o parâmetro index=False

## Avaliando e desenvolvendo intuição

Uma vez que você tiver seus dados carregados em dataframes, o Pandas faz com que uma investigação rápida dos dados seja bem simples. Vamos explorar alguns métodos úteis para avaliar e desenvolver intuição sobre um conjunto de dados. Podemos usar os dados sobre câncer que vimos anteriormente para nos ajudar.

**import** pandas **as** pd

​

df = pd.read\_csv('cancer\_data.csv')

df.head()

*# isto retorna uma tupla com as mesmas dimensões do dataframe*

*# Retornar o número de linhas e colunas do dataframe*

df.shape

*# or*

*# Retornar quantidade de registro do dataframe*

df.shape[0]

*# isto retorna os tipos de dados das colunas types datatypes list*

df.dtypes

*# embora o tipo de dados da coluna 'diagnosis' pareça ser 'object',*

*# uma investigação mais profunda revela que se trata de uma string*

type(df['diagnosis'][0])

O pandas na verdade armazena [ponteiros](https://en.wikipedia.org/wiki/Pointer_(computer_programming) para strings em dataframes e séries, e é por isso que o tipo de dados aparece como object ao invés de str. Entender este detalhe não é essencial para análise de dados - apenas saiba que strings irão aparecer como objetos no Pandas.

*# isto exibe um resumo conciso do dataframe,*

*# incluindo o número de valores não-nulos em cada coluna*

df.info()

*# isto retorna o número de valores únicos em cada coluna*

df.nunique()

*# retornar linhas com duplicidades*

df.duplicated()

*# somar as linhas duplicadas*

df.duplicated().sum()

*# Quantos valores únicos de quantidade existem no conjunto de dados sobre vinho tinto?*

*# subtrair da quantidade total!!!*

df\_r.duplicated('quality').sum()

*# OR e verificar a coluna quality*

df\_r.nunique()

*# isto retorna a média das colunas*

*#Generates descriptive statistics that summarize the central tendency, dispersion and shape of a datas# let’s distribution, excluding NaN values*

df.describe()

*# como padrão, retorna as primeiras cinco linhas*

df.head()

*# embora você possa especificar quantas linhas você gostaria que fossem retornadas*

df.head(20)

*# isso também se aplicar ao comando `.tail()` que retorna as últimas linhas do dataframe*

df.tail(2)

*# Qual é a densidade média do conjunto de dados sobre vinho tinto? Retorna a média da coluna*

df.mean()

*# Countar número de linhas sem dados que estão em branco ou vazias*

df.isnull().sum()

*#Set() Mostrar informações agrupadas de um campo – Conjunto de Dados do Campo*

set(df['Fuel'])

{'Diesel', 'Electricity', 'Ethanol/Gas', 'Gasoline', 'Gasoline/Electricity'}

## Indexando e selecionando dados no Pandas

Vamos separar este dataframe em três novos dataframes - um para cada métrica (média, desvio padrão e máximo). Para obter os dados para cada dataframe, precisamos selecionar as colunas id e diagnosis, bem como as dez colunas correspondentes àquela métrica.

*# Exibir o índice e rótulo de cada coluna*

**for** i, v **in** enumerate(df.columns):

print(i, v)

Podemos selecionar dados usando loc e iloc, cujos detalhes podem ser lidos na [documentação oficial](https://pandas.pydata.org/pandas-docs/stable/indexing.html). loc usa rótulos de linhas ou colunas para selecionar dados, enquanto iloc usa índices. Vamos usar estes para indexar o dataframe abaixo.

*# selecionar todas as colunas desde 'id' até a última coluna relacionada à média*

df\_means = df.loc[:,'id':'fractal\_dimension\_mean']

df\_means.head()

*# repita o passo acima usando índices*

df\_means = df.iloc[:,:11]

df\_means.head()

Vamos salvar o dataframe de médias para mais tarde.

df\_means.to\_csv('cancer\_data\_means.csv', index=**False**)

# Prática de limpeza

Vamos começar praticando como lidar com valores ausentes ou dados duplicados com cancer\_data\_means.csv.

*# importe pandas e carregue dados sobre cancer*

**import** pandas **as** pd

​

df = pd.read\_csv('cancer\_data\_means.csv')

*# verifique que colunas têm valores ausentes com info()*

df.info()

​

*# use médias para preencher valores ausentes*

mean = df['smoothness\_mean'].mean()

df['smoothness\_mean'] = df['smoothness\_mean'].fillna(mean)

​

*# or*

​

mean = df['texture\_mean'].mean()

df['texture\_mean'].fillna(mean,inplace = **True**)

​

mean = df['symmetry\_mean'].mean()

df['symmetry\_mean'].fillna(mean,inplace = **True**)

​

*# confirme sua correção com info()*

df.info()

​

*# verifique se há dados duplicado*

df.duplicated()

sum(df.duplicated())

*# elimine dados duplicados*

df.drop\_duplicates(inplace=True)

*# confirme correção verificando novamente se há dados duplicados*

sum(df.duplicated())

# Renomeando colunas

Já que também alteramos nosso conjunto de dados anteriormente para incluir apenas médias de características de tumores, o sufixo "\_mean" ao final do nome de cada característica parece desnecessário. Isso só faz com que mais tempo seja gasto mais tarde, na digitação da nossa análise. Vamos criar uma lista de novos rótulos para atribuir às nossas colunas.

*# remova "\_mean" dos nomes das colunas*

new\_labels = []

**for** col **in** df.columns:

**if** '\_mean' **in** col:

new\_labels.append(col[:**-**5]) *# exclua os últimos 6 caracteres*

**else**:

new\_labels.append(col)

​

*# novos rótulos para nossas colunas*

new\_labels

*# atribua novos rótulos às colunas do dataframe*

df.columns = new\_labels

*# exiba as primeiras linhas do dataframe para confirmar as alterações*

df.head()

## Renomear mais eficiente

newlabels = list(df.columns)

newlabels[6] = “new\_columns\_names”

df.columns = newlabels

# NumPy - Appending

O [NumPy](http://www.scipy.org/numpy" \t "_blank) é o pacote básico da linguagem [Python](http://www.python.org/) que permite trabalhar com arranjos, vetores e matrizes de N dimensões, de uma forma comparável e com uma sintaxe semelhante ao software proprietário [Matlab](http://www.mathworks.com/" \t "_blank), mas com muito mais eficiência, e com toda a expressividade da linguagem. Provê diversas funções e operações sofisticadas, incluindo (mas não se limitando a):

* Objeto array para a implementação de arranjos multidimensionais
* Objeto matrix para o cálculo com matrizes
* Ferramentas para álgebra linear
* Transformadas de Fourier básicas
* Ferramentas sofisticadas para geração de números aleatórios

Além disso tudo, as classes criadas podem ser facilmente herdadas, permitindo a customização do comportamento (por exemplo, dos operadores típicos de adição, subtração, multiplicação, etc.). O módulo é implementado em linguagem C, o que dá uma grande velocidade às operações realizadas.

*# Calcular tempo de execução para comparar o Numpy*

import numpy as np

import time

a = np.random.random(int(1e8))

start = time.time()

sum(a)/len(a)

print(time.time()-start, ‘seconds’)

start = time.time()

np.mean(a)

print(time.time()-start, ‘seconds’)

## Adicionando colunas

<https://chrisalbon.com/python/pandas_assign_new_column_dataframe.html>

### Anexando dados

Primeiro, importe os pacotes necessários e carregue os arquivos winequality-red.csv e winequality-white.csv.

*# importe o numpy e o pandas*

**import** pandas **as** pd

**import** numpy **as** np

​

*# carregue os conjuntos de dados de vinhos tintos e brancos*

red\_df = pd.read\_csv('winequality-red.csv',';')

white\_df = pd.read\_csv('winequality-white.csv',';')

## Crie colunas de cor

Crie dois vetores de tamanho igual ao número de linhas nos dataframes tinto e branco que repetem o valor “red” or “white”. O NumPy oferece uma forma bem fácil de fazer isso. Aqui está a documentação para a função [repeat do NumPy](https://docs.scipy.org/doc/numpy/reference/generated/numpy.repeat.html" \t "_blank). Dê uma olhada e tente você mesmo.

*# crie vetor de cor para o dataframe tinto*

color\_red = np.repeat('red', 1599) #total de linhas para repetir 1599!!

​

*# crie vetor de cor para o dataframe branco*

color\_white = np.repeat('white',4898)

​

Adicione os vetores de cor aos dataframes tinto e branco. Faça isso associando uma nova coluna chamada 'color' ao vetor apropriado. A célula abaixo faz isso para o dataframe tinto.

red\_df['color'] = color\_red

red\_df.head()

# Groupby

By “group by” we are referring to a process involving one or more of the following steps

* Splitting the data into groups based on some criteria
* Applying a function to each group independently
* Combining the results into a data structure

wine\_df.groupby([‘quality’,’color’], as\_index=False)[‘pH’].mean()

# Merge, join, and concatenate

<https://pandas.pydata.org/pandas-docs/stable/merging.html#database-style-dataframe-joining-merging>

# Função STRPTIME

<http://pubs.opengroup.org/onlinepubs/009695399/functions/strptime.html>

# Query no Pandas

Outra função útil que você usará é a função [**query do Pandas**](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.query.html).

Na aula anterior, selecionamos linhas em um dataframe indexando com uma máscara. Aqui estão os mesmos exemplos, juntamente a instruções equivalente que utilizam query().

*# selecionando registros malignos em dados de câncer*

df\_m = df[df['diagnosis'] == 'M']

df\_m = df.query('diagnosis == "M"')

*# selecionando registros de pessoas que ganham mais de $50K*

df\_a = df[df['income'] == ' >50K']

df\_a = df.query('income == " >50K"')

# Matplotlib - Plotting com o Pandas

## Explorando dados com visualização

Use o espaço abaixo para explorar o arquivo powerplant\_data\_edited.csv e responda as perguntas do teste abaixo.

*# importa e carrega dados*

**import** pandas **as** pd

​

**%** matplotlib inline

​

df = pd.read\_csv('powerplant\_data\_edited.csv')

​

df.info()

*# trace um gráfico com a relação entre temperatura e saída elétrica*

df.hist()

*# trace um gráfico com a relação entre temperatura e saída elétrica*

#df.hist(figsize=(8,8))

#df['AP'].plot(kind='hist');

*# Count ocorr*

df['V'].value\_counts();

*# Show Count by Graph Bar*

#df['V'].value\_counts().plot(kind='bar');

*# Show Count by Graph Pie*

#df['V'].value\_counts().plot(kind='pie', figsize=(8,8));

*# Display histagram*

pd.plotting.scatter\_matrix(df, figsize=(15,15));

# Another Graph

df.plot(x='compactness', y='concavity', kind='scatter');

*# another Graph*

df['AV'].plot(kind='box');

*#Another Sample*

import pandas as pd

% matplotlib inline

df = pd.read\_csv('cancer\_data.csv')

df.head()

*# Filter Diagnosis = 'M'*

df\_m = df[df['diagnosis'] == 'M']

*# Display info*

df\_m.head()

*# Filter Diagnosis = 'M'*

mask = df['diagnosis'] == 'M'

*# Display info*

df\_m = df[mask]

df\_m

*# Filter*

df\_b = df[df['diagnosis'] == 'M']

df\_b['area\_max'].describe()

'{0:.4f} while that of benign tumors is {1:.4f}'.format(df\_m['area\_max'].mean(), df\_b['area\_max'].mean());

## Criando um gráfico de barras usando matplotlib

**import** matplotlib.pyplot **as** plt

**%** matplotlib inline

Dois argumentos são necessários para se usar a função bar do pyplot: a coordenada no eixo x das barras e sua altura.

plt.bar([1, 2, 3], [224, 620, 425]);

Você pode especificar os rótulos das marcações do eixo x usando a função xticks do pyplot, ou ainda especificando um parâmetro adicional na função bar. As duas células abaixo The two cells below accomplish the same thing.

*# trace as barras*

plt.bar([1, 2, 3], [224, 620, 425])

​

*# especifique as coordenadas no eixo x das marcações e seus rótulos*

plt.xticks([1, 2, 3], ['a', 'b', 'c']);

*# trace as barras com rótulos nas marcações do eixo x*

plt.bar([1, 2, 3], [224, 620, 425], tick\_label=['a', 'b', 'c']);

*#Defina o título e o rótulo dos eixos assim.*

plt.bar([1, 2, 3], [224, 620, 425], tick\_label=['a', 'b', 'c'])

plt.title('Some Title')

plt.xlabel('Some X Label')

plt.ylabel('Some Y Label');

*# Crie um gráfico de barras com rótulos adequados*

*# e Group by agrupando informações:*

df.groupby(['acidity\_levels'])['pH','quality'].mean().plot(kind='bar')



## Traçando gráficos com matplotlib

Use matplotlib para criar gráficos de barras que permitam visualizar as conclusões que você tirou com groupby e query.

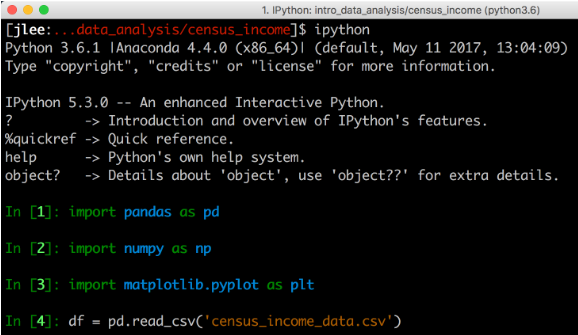
<https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot>

# IPython – Command Line

O **[IPython](http.s://ipython.org/" \t "_blank)** é, na verdade, quem fornece o kernel Python interativo que usamos no Jupyter Notebook. E, de fato, podemos usar o IPython fora do Jupyter Notebook com sua interface de linha de comando no nosso terminal. Isso é muito conveniente e demais para fazer modificações rápidas, exploração, experimentação e até executar scripts em Python!

Leia mais sobre como o IPython e o Jupyter Notebook funcionam [**aqui**](http://jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html). [**Aqui**](http://ipython.readthedocs.io/en/stable/) você ainda encontra a documentação oficial do IPython.

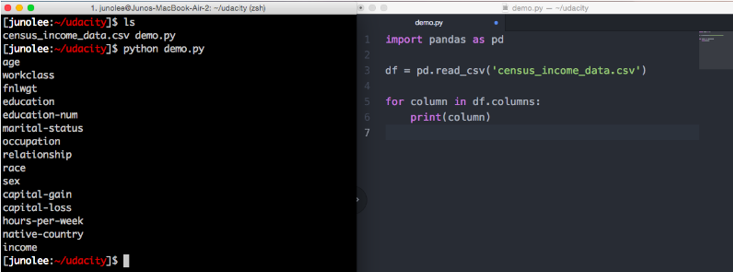
Para usar a linha de comando, basta digitar “ipython” no seu terminal. Se você já tiver o Jupyter Notebook instalado, isso deve funcionar. Como sempre fazemos no Jupyter Notebook, vamos importar os pacotes e carregar em um conjunto de dados.



## Criando scripts com análises

Quando falamos de tarefas e projetos de programação, ser capaz de criar e executar scripts não tem preço.

Você pode escrever o código em um editor de texto e executar o arquivo no terminal. Aqui temos um exemplo simples que exibe os nomes de coluna do conjunto de dados de renda do censo. Se você salvar o arquivo em formato .py com o código exibido à direita, poderá executá-lo conforme mostrado à esquerda. Tenha certeza de que você está no diretório em que salvou o arquivo!



## Run Python File

%run -i loadLibraryDefault.py

# Referências:

EPA - Data on Cars used for Testing Fuel Economy

<https://www.epa.gov/compliance-and-fuel-economy-data/data-cars-used-testing-fuel-economy>

DOWNLOAD

<http://www.fueleconomy.gov/feg/download.shtml/>

# The Python Standard Library

While [The Python Language Reference](https://docs.python.org/3/reference/index.html#reference-index) describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

<https://docs.python.org/3/library/>

## 9.2. [math](https://docs.python.org/3.6/library/math.html?highlight=math%20module" \l "module-math" \o "math: Mathematical functions (sin() etc.).) — Mathematical functions

Referencia biblioteca Python

https://docs.python.org/3.6/library/math.html?highlight=math%20module#module-math

# Pendências FORUM

Semana 3 - Aula 2 – 15 Conclusões usando Groupby – 22/12/2017

Não postado, mas com dúvidas:

Semana 3 - Aula 2 – 19 Plotting com Matplotlib – 22/12/2017

# Use groupby para obter a qualidade média para cada nível de acidez

# Crie um gráfico de barras com rótulos adequados

Você pode usar a função [plot](https://matplotlib.org/api/pyplot_api.html" \l "matplotlib.pyplot.plot" \t "_blank) do pyplot para isso.