BIG DATA

LABDATA

Introdução ao Big Data

Tema da Aula: **Big Data com Python**

Prof.: **Dino Magri**

Data: **06 de Outubro de 2017**

Coordenação:

Prof. Dr. Adolpho Walter Pimazzi Canton

Profa. Dra. Alessandra de Ávila Montini

LABDATA

Coordenação:

Prof. Dr. Adolpho Walter Pimazzi Canton

Profa. Dra. Alessandra de Ávila Montini

- Contatos:
  - E-mail: professor.dinomagri@gmail.com
  - Twitter: https://twitter.com/prof_dinomagri
  - LinkedIn:http://www.linkedin.com/in/dinomagri
  - Site: http://www.dinomagri.com

Currículo

- **(2014-Presente)** – Professor no curso de Extensão, Pós e MBA na Fundação Instituto de Administração (FIA) – www.fia.com.br
- **(2013-Presente)** – Pesquisa e Desenvolvimento no Laboratório de Arquitetura e Redes de Computadores (LARC) na Universidade de São Paulo – www.larc.usp.br
- **(2013)** – Professor no MBA em Desenvolvimento de Inovações Tecnológicas para WEB na IMED Passo Fundo – RS – www.imed.edu.br
- **(2012)** – Bacharel em Ciência da Computação pela Universidade do Estado de Santa Catarina (UDESC) – www.cct.udesc.br
- **(2009/2010)** – Pesquisador e Desenvolvedor no Centro de Computação Gráfica – Guimarães – Portugal – www.ccg.pt
- **Lattes:** http://lattes.cnpq.br/5673884504184733

Material das aulas

- Material das aulas:
  - https://bitly.com/posmba-turma4
  - Senha: fia2017
- Faça o download do arquivo **2017-10-06-py-aula3.zip**
- Salve na Área de Trabalho (Desktop)
- Depois que finalizar o download, acesse a pasta Área de trabalho e descompacte o arquivo `2017-10-06-py-aula3.zip`

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- Python para Big Data
- Pandas
- Exemplo prático
- Exercícios

Conteúdo da Aula

- **Objetivo**
- Exercício de Revisão
- Python para Big Data
- Pandas
- Exemplo prático
- Exercícios

Objetivo

- O objetivo dessa aula é revisar os conceitos aprendidos com um **exemplo prático** e apresentar as bibliotecas do **Python para Big Data**, bem como apresentar os conceitos da biblioteca **Pandas** para **manipular** e **explorar** dados.

Conteúdo da Aula

- Objetivo
- **Exercício de Revisão**
- Python para Big Data
- Pandas
- Exemplo prático
- Exercícios

Exercício de Revisão

- Iremos construir um **módulo** em Python para recuperar as palavras da letra de uma músicas disponível em um servidor remoto.
- A proposta desse exercício é revisar os conceitos de **módulos**, **funções**, **laços de repetição** e **estruturas de controle**.
- Também iremos aprender novos conceitos ☺

Exercício de Revisão

- Para desenvolver e testar esse módulo iremos utilizar a música Aquarela, escrita por Toquinho, disponível em: http://dinomagri.com/data/aquarela.txt
- Para resolver esse exercício temos que:
  1. Recuperar e tratar os dados
  2. Criar uma função chamada `recuperar_palavras()`
  3. Criar o atributo especial `__name__`
  4. Melhorar o código desenvolvido (Refactoring)
  5. Documentar o módulo criado
  6. Definir o ambiente de execução

Ante de criar nosso módulo, vamos realizar alguns testes. Abra o notebook **aula3-parte1-exercício.ipynb**

1. Recuperar e tratar os dados

- Para recuperar a letra da música iremos utilizar a função `urlopen` do módulo `urllib.request`
  - **`urllib.request`** - esse módulo define funções e classes que possibilitam abrir URLs (na maioria das vezes HTTP).
  - **`urlopen`** – Como o nome sugere abre um URL no formato de string.
- Também iremos utilizar o comando **`with`**, que garante o fechamento da conexão caso ocorra ou não algum erro.
  - No comando **`open`** é preciso explicitamente fechar o arquivo.

1. Recuperar e tratar os dados

- Para o desenvolvimento desse exercício, vamos utilizar o Editor de Texto Sublime.
  - Caso não tenha instalado, acesse: https://www.sublimetext.com/3
  - É possível utilizar qualquer outro ambiente de desenvolvimento ou editor que suporte Python.
- Depois de abrir o editor (Sublime), crie um novo arquivo e salve na pasta **`Desktop/2017-10-06-py-aula3`** com o nome **`palavras.py`**
- Com o arquivo criado e salvo, iremos adicionar o seguinte código:

1. Recuperar e tratar os dados

```
from urllib.request import urlopen


with urlopen('http://dinomagri.com/data/aquarela.txt') as musica:
        palavras_musica = []
        for linha in musica:
                palavras_linha = linha.decode('utf-8').split()
                for palavra in palavras_linha:
                        palavras_musica.append(palavra)
```

1. Recuperar e tratar os dados

```
from urllib.request import urlopen


with urlopen('http://dinomagri.com/data/aquarela.txt') as musica:
        palavras_musica = []
        for linha in musica:
                palavras_linha = linha.decode('utf-8').split()
                for palavra in palavras_linha:
                        palavras_musica.append(palavra)
for palavra in palavras_musica:
        print(palavras)
```

1. Recuperar e tratar os dados

- Esse módulo também pode ser testado no ambiente iterativo do Python.
- Para isso, abra o CMD ou Terminal, e digite a palavra **python**
- O interpretador Python será carregado, possibilitando utilizar o módulo criado.

```
C:\Users\Dino Magri> cd Desktop/2017-10-06-py-aula3

C:\Users\Dino Magri\Desktop\2017-10-06-py-aula3> python

Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 15:51:26) [MSC v.1900 32
bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>>

>>> import palavras
```

A extensão .py foi omitida

2. Criar uma função chamada `recuperar_palavras()`

- Para ter mais controle sobre a forma de execução do nosso código, temos que criar uma função chamada `recuperar_palavras()`.
- Para isso, acesse novamente o editor de texto e adicione as partes do código (recuperar a música, criar a lista com as palavras e a impressão) dentro dessa função.
- O seguinte código deve estar presente no arquivo `palavras.py`

2. Criar uma função chamada `recuperar_palavras()`

```
from urllib.request import urlopen

def recuperar_palavras():
    with urlopen('http://dinomagri.com/data/aquarela.txt') as musica:
        palavras_musica = []
        for linha in musica:
            palavras_linha = linha.decode('utf-8').split()
            for palavra in palavras_linha:
                palavras_musica.append(palavra)
    for palavra in palavras_musica:
        print(palavra)
```

2. Criar uma função chamada `recuperar_ palavras()`

- Como vimos, podemos executar a função criada dentro do módulo de diferente maneiras:
  - Importando o módulo e chamando a função:

```
>>> import palavras
>>> palavras.recuperar_ palavras()
```

  - Importar a função diretamente do módulo:

```
>>> from palavras import recuperar_palavras
>>> recuperar_palavras()
```

2. Criar uma função chamada `recuperar_ palavras()`

- O que acontece se executarmos o código diretamente do CMD ou Terminal (sem estar no ambiente iterativo)? Abra o CMD ou terminal, acesse a pasta e execute:

```
C:\Users\Dino Magri> cd Desktop/2017-10-06-py-aula3

C:\Users\Dino Magri\Desktop\2017-10-06-py-aula3> python palavras.py
```

- **Qual foi a saída?**
- Não teve, pois esse código define a função e finaliza o script.
- Para que seja possível executar o código tanto no ambiente iterativo, e rodar como script é necessário utilizar um **atributo especial**.

3. Criar o atributo especial `__name__`

- Os atributos especiais no Python são criados por sublinhados duplos (__).
- O atributo especial **`__name__`** permite verificar se o módulo está rodando por si mesmo (em caso de script) ou se está sendo importado por outro módulo.
- Para testar esse comportamento, podemos adicionar em nosso código a seguinte linha no final.

```
print(__name__)
```

- Teste o código via ambiente iterativo e execução do script.
- **Qual foi a saída?**

3. Criar o atributo especial `__name__`

- Na execução do ambiente iterativo, quando carregamos o módulo pela primeira vez, ele imprimir o nome do módulo.
- Por outro lado, na execução via script (`python palavras.py`) ele retorna o valor **`__main__`**, que é o nome do escopo no qual o código executa. Podemos dizer que é o programa principal.
- Desta forma, podemos definir que se o valor do `__name__` for igual ao `__main__` devemos executar a função `recuperar_palavras()`.

```
if __name__ == "__main__":
    recuperar_palavras()
```

- Com isso, nosso código funciona tanto como módulo ou como script.

3. Criar o atributo especial `__name__`

- Mais qual a diferença entre módulo, script e programa?

Módulo Python
Convenientemente utilizado com APIs

Script Python
Convenientemente executado da linha de comando

Programas Python

Talvez composto por diferentes módulos

3. Criar o atributo especial `__name__`

- Mais qual a diferença entre módulo, script e programa?

4. Melhorar o código desenvolvido (Refactoring)

- Temos que:
  - Criar uma função `imprimir_items` para imprimir as palavras.
  - Criar uma função principal, chamada `main()`.
  - Na função `recuperar_palavras`, vamos passar a URL por parâmetro.
  - Utilizar o módulo `sys` para receber a URL por parâmetro na linha de comando.
  - Testar tudo ☺

5. Documentar o módulo criado

- Por fim, iremos documentar nosso código utilizando uma funcionalidade chamada **docstrings**.
  - Elas são strings que inserimos dentro do código Python com o objetivo de fornecer uma explicação sobre o funcionamento deste código.
  - Essa string deve ser colocada como a primeira linha da definição de uma classe, método, função ou módulo.
- Utiliza-se três aspas duplas (" " ") para abrir e fechar a documentação.
- É recomendado utilizar o Google Python Style Guide
  - https://google.github.io/styleguide/pyguide.html

5. Documentar o módulo criado

- Basicamente temos que adicionar na primeira linha de cada função, a seguinte string:

Adicione essa *docstring* no arquivo `palavras.py` e complete a documentação das outras funções e do módulo

```
def recuperar_palavras(url):
        """Recupera uma lista de palavras de uma URL.
        Args:
                url: A URL de um documento no formato UTF-8.
        Returns:
                Uma lista de strings contendo as palavras do documento.
        """
```

6. Definir o ambiente de execução

- Para finalizar, é comum em sistema baseado em Unix (Linux, MacOS, BSD) utilizar os caracteres `#!` para definir qual interpretador deve ser utilizado para executar o código.
- Para isso, podemos adicionar na primeira linha o seguinte comentário especial:

```
#!/usr/bin/env python3
```

- Note que se estiver no Windows, o Python irá identificar e tratar isso tranquilamente.
- Utilizando esse comentário especial, não tem a necessidade de digitar a palavra python antes de rodar o código:

```
~~python~~ palavras.py http://dinomagri.com/data/aquarela.txt
```

#! é chamado de *shebang ou hash-bang – Fonte:* https://en.wikipedia.org/wiki/Shebang_(Unix)

6. Definir o ambiente de execução

- Se utilizar Linux ou Mac OS, será necessário rodar o comando `chmod +x` para permitir a execução do script e a utilização do interpretador definido na primeira linha.

```
$ chmod +x palavras.py
$ ./palavras.py http://dinomagri.com/data/aquarela.txt
```

Exercício de Revisão

- **Resumindo o que fizemos:**
  - Criamos arquivos `*.py` chamados "módulos".
  - Os módulos podem ser executado diretamente como script ...

    ```
    python nome_modulo.py
    ```
  - ... ou importados no ambiente iterativo

    ```
    import nome_modulo
    ```
  - Utilizamos o `__name__` para determinar como o módulo é utilizado.
  - O módulo `sys` foi utilizado para passar o parâmetro (URL) via linha de comando.
  - Documentamos o nosso código com **docstrings**
  - Aprendemos como definir qual interpretador deve controlar a execução do módulo utilizando o comentário especial (`#!`).

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- **Python para Big Data**
- Pandas
- Exemplo prático
- Exercícios

Introdução

- Como vimos, Python é uma linguagem muito poderosa, simples e fácil de utilizar.
- Em Python existem diversas bibliotecas disponíveis para Big Data em áreas como:
  - Data Quality e Data Preparation
  - Data Mining
  - Aprendizagem de Máquina
  - Matemática e Estatística
  - Processamento de linguagem natural
  - Visualização
  - Entre outras ...

Introdução

- As principais bibliotecas que destaco para cada área são:
  - Bibliotecas fundamentais para Computação Científica
    - IPython Notebook, Numpy, Pandas, SciPy
  - Matemática e Estatística
    - Statsmodels e SymPy
  - Aprendizagem de Máquina
    - Scikit-learn, TensorFlow e Theano
  - Visualização e Plotagem
    - Matplotlib, Bokeh, Seaborn, Plotly, Basemap, NetworkX
  - Biblioteca para Data Mining e Processamento de Linguagem Natural
    - Scrapy, NLTK, Pattern e Gensim

Antes de continuar ....

- Para rodar todos os exemplos das bibliotecas no **Windows 64 bits**, será necessário instalar os seguintes softwares:
  - **Visual C++** 2015 Build Tools - http://landinghub.visualstudio.com/visual-cpp-build-tools
  - **NumPy** com suporte a biblioteca Intel Math Kernel que inclui as DLLs necessárias no diretório principal do numpy - http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy
  - **Scipy** é um software para matemática, ciência e engenharia. É a base da grande maioria das bibliotecas - http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy
- Faça o download desses arquivos em relação a versão do Python, por exemplo, para a versão **3.6** do Python, faça o download das bibliotecas:
  - numpy-1.13.1+mkl-**cp36-cp36m**-win_amd64.whl
  - scipy-0.19.1-**cp36-cp36m**-win_amd64.whl

Se a versão do Python for 3.5, realize o download dessas bibliotecas na versão cp35-cp35m

Antes de continuar ....

- Instale primeiro o Visual C++ 2015 Build Tools
- Abra o CMD e acesse a pasta onde salvou os arquivos (e.g. Downloads)

```
C:\Users\Dino Magri>

C:\Users\Dino Magri>  cd Downloads
```

- Realize a instalação primeiro do NumPy

```
C:\Users\Dino Magri\Downloads> pip3 install numpy-1.13.1+mkl-cp36-cp36m-win_amd64.whl
```

- Depois realize a instalação do SciPy

```
C:\Users\Dino Magri\Downloads> pip3 install scipy-0.19.1-cp36-cp36m-win_amd64.whl
```

Antes de continuar ....

- Por fim, realize a instalação das seguintes bibliotecas:

```
C:\Users\Dino Magri\Downloads> pip3 install pandas statsmodels scikit-learn
matplotlib bokeh urllib3 xlrd
```

- **Nota**: Os notebooks com exemplos dessas bibliotecas, foram testados no Windows 64 bits.
  - Os códigos funcionam no Linux e MacOS, porém a instalação das bibliotecas e pacotes utilizados podem variar de sistema operacional para sistema operacional.

- Para realizar o download dos arquivos necessários para rodar os exemplos demonstrados nessa aula, acesse: http://bit.ly/bibliotecas-python

Numpy

http://numpy.org

- Numpy é uma biblioteca fundamental para computação científica com Python. Suas principais funcionalidades:
  - Array de n-dimensões
  - Funções sofisticadas para trabalhar com esses arrays
  - Ferramentas para integrar código C/C++ e Fortran
  - Útil para álgebra linear, transformada de Fourier e capacidade de gerar números aleatórios
- Foi desenvolvido em 2005 e hoje é a base de outras bibliotecas como Pandas e Scikit-learn.

Mais detalhes sobre essa biblioteca pode ser visualizado no notebook: **aula3-parte2-numpy.ipynb**

Pandas

http://pandas.pydata.org/

- Pandas é uma biblioteca de código aberto para análise de dados em Python.
- Foi desenvolvido em 2008 por Wes McKinney.
  - Tem uma grande comunidade - http://pandas.pydata.org/community.html
  - Melhorias continuas - https://github.com/pydata/pandas/
  - Diversas funcionalidades
  - Iteração rápida
- Essa biblioteca teve uma **ótima adoção**, se tornando a **biblioteca padrão para análise de dados** utilizando Python.

Mais detalhes sobre essa biblioteca será visto nesta aula.

Statsmodels

StatsModels
Statistics in Python
http://www.statsmodels.org

- É um biblioteca Python que fornece classes e funções para estimativa de diversas funções estatísticas.
  - É útil para realizar testes como ANOVA, ARMA, **Series tempo**rais, diversos tipos de regressão.
  - Resultados são testados em razão de pacotes de estatísticas existentes para garantir a precisão.
- Os módulos são originalmente do `scipy.stats` e foi inicialmente escrito por Jonathan Taylor.
- Como parte do *Google Summer of Code 2009*, o **statsmodels** foi testado, melhorado e disponibilizado como pacote.
  - Desde então conta com o suporte do Google e AWR para o desenvolvimento.

Statsmodels

StatsModels
Statistics in Python
http://www.statsmodels.org

- Algumas das funcionalidades do pacote `statsmodels` incluem:
  - Diversos modelos de regressão
  - Estatística descritiva
  - Testes estatísticos
  - Análise de series temporais
  - Entre outros

! Mais detalhes sobre essa biblioteca pode ser visualizado no notebook: **aula3-parte2-statsmodel.ipynb**

Scikit-learn

http://scikit-learn.org/

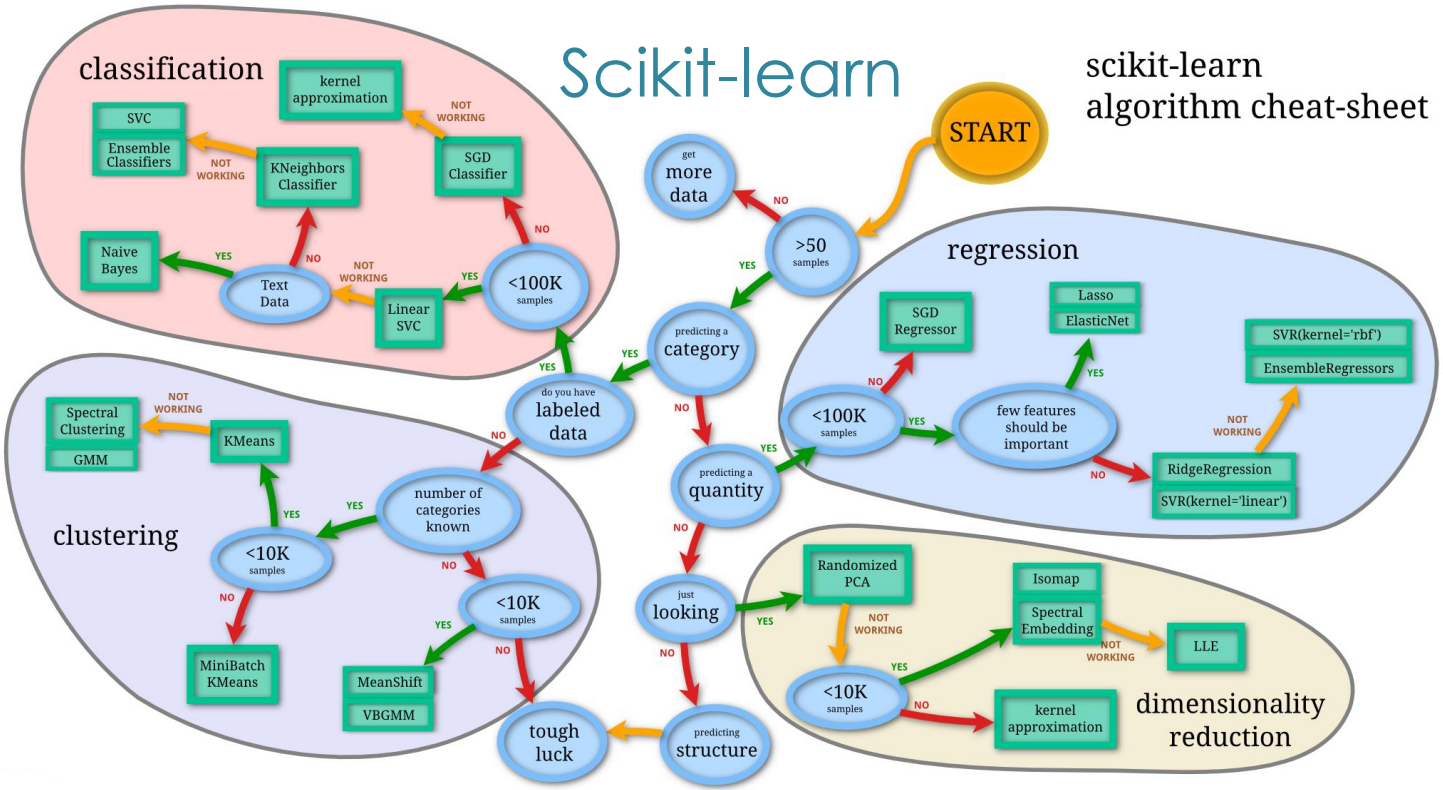
- É uma biblioteca para aprendizado de máquina em Python, que tem como principal foco:
  - Fornecer ferramentas **eficientes** e **simples** para Data Mining e Data Analysis.
  - Acessível para todos e **reutilizável em vários contextos**.
  - Sua construção faz uso do NumPy, SciPy e matplotlib
- Provavelmente um dos melhores frameworks de propósito geral de aprendizado de máquina.
- Iniciou como sendo um projeto do *Google Summer of Code* em 2007 por David Cournapeau, e foi utilizado na tese de Matthieu Brucher.

Scikit-learn

http://scikit-learn.org/

- Em 2010, INRIA disponibilizou a primeira versão, e financiou o projeto junto com o Google, Tinyclues e a Python Software Foundation.
- Muitas empresas utilizam o **scikit-lean**, alguns exemplos:
  - Spotify, Evernote, Google, Data Robot
- As principais características:
  - Modelos lineares generalizados
  - SVMs, kNN, *Bayes*, *Decision Trees*, *Ensembles*
  - Algoritmos de *Clustering* e *Density*
  - Validação cruzada, *Pipelining*, Avaliação dos modelos
  - Transformações dos conjuntos de dados
  - Entre outras

Scikit-learn



http://scikit-learn.org/

! Mais detalhes acesse o notebook: **aula3-parte2-scikit-learn.ipynb**

TensorFlow

https://www.tensorflow.org/

- É uma biblioteca que realiza a computação numérica como um grafo.
- Os nós do grafo são operações que tem qualquer número de entradas e saídas.
- As arestas do grafo são tensores (tensor) que flui (flow) entre os nós e representam dados em arrays multidimensionais.
- A arquitetura flexível permite que a computação seja realizada tanto na CPU como na GPU.
- Foi desenvolvido por um time interno do Google e lançado em novembro de 2015.

matplotlib

matplotlib
https://matplotlib.org/

- É uma das bibliotecas de visualização mais antiga do Python (2002), porém muito utilizada ainda.
- Funciona muito bem para realizarmos análises iniciais no dados.
- Ela é muito poderosa, porém complexa!
- Um dos pontos mais criticados é o estilo padrão, que passa uma sensação dos anos 90. Porém a versão 2.0 terá algumas melhorias nesse sentido.
  - Galeria de exemplos: http://matplotlib.org/examples/index.html

Mais detalhes sobre essa biblioteca será visto nas próximas aulas.

bokeh

Bokeh

https://bokeh.pydata.org/

- É uma biblioteca de **visualização interativa** que roda nos principais navegadores.
- Foi criado pela Continuum Analytics.
- Possibilita construir de **forma simples** e **rápida** gráficos **elegantes**, concisos e de alto desempenho em grandes volumes de dados.
  - Galeria de exemplos: https://bokeh.pydata.org/en/latest/docs/gallery.html

Mais detalhes sobre essa biblioteca será visto nas próximas aulas.

NetworkX

NetworkX
https://networkx.github.io/

- É um pacote Python para criação, manipulação e estudo de estruturas, da dinâmica e funções de redes complexas.
- Características:
  - Estruturas de dados para grafos, digrafos e multigrafos
  - Muitos algoritmos de grafos
  - Estrutura da rede e medidas de análises
  - Geradores para gráficos clássicos, aleatórios e redes sintéticas.
  - Os nós podem ser "qualquer coisa" (texto, imagem, XML, etc).
  - Arestas podem armazenar dados arbitrários (por exemplo, pesos, séries temporais)
- Originalmente foi desenvolvida por Aric Hagberg, Pieter Swart, Dan Schult.
- A versão inicial foi disponibilizada em 2005.

NLTK

NLTK
http://www.nltk.org

- É uma plataforma para criação de programas Python para trabalhar com dados de linguagem humana.
- Ele fornece interfaces fáceis de usar para mais de 50 corpora e recursos léxicos, além é claro de conter um conjunto de ferramentas para processar o texto, como:
  - Classificação, tokenização, derivação, marcação, análise e raciocínio semântico.
- Os autores originais foram Steven Bird, Edward Loper, Ewan Klein.
- A primeira versão foi publicada em 2001.

NLTK

NLTK
http://www.nltk.org

- Principais aplicações:
  - Análise de sentimento
  - Filtragem de spam
  - Detectar plágio
  - Similaridade de documentos
  - Categorizar documentos
  - Realizar pesquisa inteligente
  - Análise de frequência de palavras
  - Entre outros

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- Python para Big Data
- **Pandas**
- Exemplo prático
- Exercícios

Pandas

- Pandas é uma biblioteca de código aberto para análise de dados em Python.
- Foi desenvolvido em 2008 por Wes McKinney.
  - Tem uma grande comunidade - http://pandas.pydata.org/community.html
  - Melhorias continuas - https://github.com/pydata/pandas/
  - Diversas funcionalidades
  - Iteração rápida
- Essa biblioteca com o tempo teve uma **ótima adoção**, se tornando a **biblioteca padrão para análise de dados** utilizando Python.

Pandas

- Principais características do Pandas
  - É possível **processar diversos conjuntos de dados em diferentes formatos** (Series temporais, dados tabulares heterogêneos, e matrizes)
  - Facilidade de **importar dados** de diversas fontes como CSV, DB/SQL.
  - Podemos lidar com **diversas operações** nesses conjuntos de dados: filtragem, agrupamento, reordenamento, remodelação, junção, fatiamento, entre outros.
  - Facilita trabalhar com **dados** que estão **faltando**.
  - Tem uma boa integração com outras bibliotecas Python, como a statsmodels, SciPy, e scikit-lean.

Pandas

- Benefícios de utilizar pandas
  - Representação dos dados
  - Filtragem e fatiamento dos dados
  - Código limpo e conciso
  - **Bom desempenho**, especialmente quando faz computação numérica, afinal ele foi construído em cima da biblioteca NumPy.

Pandas

- Instalação
  - `pip install pandas`
    - `pytz` – biblioteca para calculo de `timezone`.
    - `numpy` – processamento de array numéricos.
    - `python-dateutil` – fornece extensão para o módulo de `datetime`.
    - `six` – fornece funções que permitem diminuir as diferenças entre as versões do Python 2 e 3.

Pandas

- Conforme vimos Pandas foi construído em cima do NumPy e ele fornece diversas outras funcionalidades que não estão disponíveis no NumPy.
- **Permite criar estruturas de dados de fácil entendimento e que são rápidas**.
- Desta forma possibilita preencher a lacuna que existia entre Python e linguagem como R.

Pandas

- As 3 principais estruturas de dados no Pandas:
  - **Series**, **DataFrame** e Panel (não será abordado!)

- Para fazer uso dessas estruturas, primeiro precisamos importar a biblioteca.

```
>>> import pandas as pd
```

Pandas - Series

- **Series** é na verdade um array NumPy de 1 dimensão **com rótulos**.
- Podemos criar Series da seguinte maneira:

```
>>> s = pd.Series(dados)
```

- Onde, `dados` pode ser um dos itens abaixo:
  - Um `numpy.ndarray`
  - Um `dicionário`
  - Um `valor escalar`

Pandas - Series

- Além da criação, podemos realizar operações como **fatiamento** (slicing), **atribuições**, aplicar funções aritméticas e estatísticas, entre tantos outros.

Abra o arquivo "aula3-parte3-series.ipynb"

Pandas - DataFrame

- **DataFrame** é um array 2D com rótulos nas colunas e nas linhas.
- Conceitualmente é semelhante a uma tabela ou planilha de dados.
- Tem as seguintes características:
  - Colunas podem ser de diferentes tipos: `float64, int, bool`.
  - Uma coluna do DataFrame é uma `Series`.
  - Podemos pensar que é um dicionário de Series, onde as colunas e linhas são indexadas, denota-se `index` para linhas e `columns` no caso de colunas.
  - Os índices são necessários para acesso rápido aos dados.
  - Seu **tamanho** é **mutável**: colunas podem ser inseridas e deletadas

Pandas - DataFrame

- Podemos criar DataFrame da seguinte maneira:

```
>>> df = pd.DataFrame(dados)
```

- Onde, `dados` pode ser:
  - Dicionário de ndarrays de 1D, listas, dicionários, ou Series
  - Array 2D do NumPy
  - Estruturado
  - Series
  - Outra estrutura DataFrame

Pandas - DataFrame

- Podemos realizar inúmeras operações, como **seleção**, **atribuição**, **remoção**, alinhamento, aplicar funções aritméticas e estatísticas entre outros.

Abra o arquivo "aula3-parte3-dataframe.ipynb"

Outras operações

- Conforme vimos o fatiamento e a indexação podem ser um pouco confusos.
  - Por exemplo, se uma Series tem um índice explicito de inteiros, uma operação como `s1[1]` irá utilizar o índice **explícito**, enquanto que uma operação de fatiamento como `s1[1:3]` irá utilizar o índice **implícito**, no mesmo estilo de Python. Exemplo:

```
>>> s1 = pd.Series(['a', 'b', 'c'], index=[1,3,5])
1    a
3    b
5    c
dtype: object
```

Outras operações

- Utilizando o índice **explicito** quando se está indexando irá produzir o resultado:

```
>>> s1[1]
'a'
```

- Utilizando o índice **implícito** quando se está fatiando irá produzir o resultado:

```
>>> s1[1:3]
3    b
5    c
dtype: object
```

Outras operações

- Como podemos perceber isso pode ser um pouco confuso no caso de índices de números inteiros.
- Por isso, Pandas fornece alguns **indexadores especiais** que explicitamente contém esquemas de acesso aos índices.
- Eles não são métodos funcionais e sim **atributos que expõe uma interface de fatiamento particular para o dados**.
- São eles: `loc`, `iloc`

Outras operações

- O atributo `loc` permite indexar e fatiar sempre utilizando o índice explícito.

```
>>> s1
1    a
3    b
5    c
>>> s1.loc[1]
'a'
>>> s1.loc[1:3]
1    a
3    b
```

Outras operações

- O atributo `loc` permite indexar e fatiar sempre utilizando o índice implícito.

```
>>> s1
1    a
3    b
5    c
>>> s1.iloc[1]
'b'
>>> s1.iloc[1:3]
3    b
5    c
```

Abra o arquivo "aula3-parte3-dataframe.ipynb"

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- Python para Big Data
- Pandas
- **Exemplo prático**
- Exercícios

Exemplo prático

- Iremos utilizar o arquivo `capitais.csv` que é um arquivo que tem todas as capitais do Brasil, bem como a população e a área de cada capital (km2).
- O Pandas disponibiliza diversos métodos para carregar diferentes tipos de dados, segue alguns deles:
  - `pd.read_csv('caminho-ate-arquivo.csv', sep=';')`
  - `pd.read_excel('caminho-ate-arquivo.xlsx', 'Sheet1')`
  - `sql.read_frame(query, connection)` – necessita do módulo pandas.io

Abra o arquivo "aula3-parte4-exemplo-pratico.ipynb"

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- Python para Big Data
- Pandas
- Exemplo prático
- **Exercícios**

Exercícios

Utilizando o dataframe `capitais` criado anteriormente, faça os seguintes exercícios:

- **Exercício 1** - Selecione todas as capitais que tenham área maior que 400 km2. Quantas foram?

- **Exercício 2** - Selecione as capitais que tenham população maior que 2 milhões.

Exercícios

- **Exercício 3** - Crie uma função que retorne uma lista contendo somente as capitais que começam com uma determinada letra. A função deve receber dois parâmetros:
  - O primeiro parâmetro será uma lista com **todas as capitais**.
  - O segundo será uma **letra**.
- Para testar a função, selecione as capitais que começam com as letras **B** e **Z**. Lembre-se de tratar possíveis erros.

```
>>> def capitais_com_letra(todas_capitais, letra):
```

Exercícios

- **Exercício 4** - Utilizando a função criada no exercício 3, calcule o total da população para as capitais que começam com **S**. Por fim, imprima a seguinte frase:

```
As capitais X, Y e Z tem W pessoas.
```

Referências Bibliográficas

- **Mastering pandas** – Femi Anthony – Packt Publishing, 2015.
- **Python for Data Analysis** – Wes McKinney – USA: O'Reilly, 2013.
- Tutoriais disponíveis no site oficial do Pandas - http://pandas.pydata.org/pandas-docs/version/0.18.0/tutorials.html
- Livro de receitas disponíveis no site oficial do Pandas - http://pandas.pydata.org/pandas-docs/version/0.18.0/cookbook.html

Referências Bibliográficas

- **Python for kids – A playful Introduction to programming** – Jason R. Briggs – San Francisco – CA: No Starch Press, 2013.

- **Python Cookbook** – David Beazley & Brian K. Jones – O'Reilly, 3th Edition, 2013.

- As referências de links utilizados podem ser visualizados em http://urls.dinomagri.com/refs