BIG DATA

LABDATA

FUNDAÇÃO INSTITUTO DE ADMINISTRAÇÃO

Introdução ao Big Data

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

Prof.: **Dino Magri**

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

Coordenação:

Prof. Dr. Adolpho Walter Pimazzi Canton

Profa. Dra. Alessandra de Ávila Montini

Currículo

- **(2014-Presente)** – Professor no curso de Extensão, Pós e MBA na Fundação Instituto de Administração (FIA) – www.fia.com.br
- **(2018-Presente)** – Pesquisa e Desenvolvimento de Big Data e Machine Learning na Beholder (http://beholder.tech)
- **(2013-2018)** – Pesquisa e Desenvolvimento no Laboratório de Arquitetura e Redes de Computadores (LARC) na Universidade de São Paulo – www.larc.usp.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

- Caso esteja utilizando seu próprio computador, realize o download de todos os arquivos e salve na **Área de Trabalho** para facilitar o acesso.
  - Lembre-se de instalar os softwares necessários conforme descrito no documento de Instalação (**`InstalaçãoPython3v1.2.pdf`**).

- Nos computadores da FIA os arquivos já estão disponíveis, bem como a instalação dos softwares necessários.

Conteúdo da Aula

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

Conteúdo da Aula

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

Objetivo

- O objetivo dessa aula é revisar os conceitos aprendidos, apresentar algumas 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**
- Novos conceitos
- Python para Big Data
- Pandas
- Exemplo prático
- Exercícios

Argumentos

</> Abra o arquivo **"aula3-parte1-exercicios-revisao.ipynb"**

Exercício 1

- Considere a frase: `"Programando em Python na FIA!!!"`.
  - a) Escreva um programa que realize a **contagem das letras maiúsculas, minúsculas e caracteres especiais**.
  - b) Realize a contagem de caracteres existentes na mesma frase
- Utilize um dicionário para salvar a quantidade de vezes que cada caractere aparece. Utilize estruturas de repetição e controle.

Exercício 2

- Escreva um programa para imprimir apenas o caracteres (a, e, i, o, u) da seguinte frase `"Estou programando em Python"`.
  - É necessário tratar caso a letra seja maiúscula e minúscula.
  - Utilize estruturas de repetição e controle.

Exercício 3

- Escreva um programa em Python que irá somar todos os elementos numéricos da lista.
- Utilize as seguintes listas para resolver o exercício:
  - `lista3 = [6, 1.5, 2, -8, 20, 1.23]`
  - `lista4 = [6, 1.5, 2, "fia", -8, 20, 1.23, [1,2]]`

Exercício 4

- Crie uma função com o nome `maior_str` que receba dois textos (strings) por parâmetro e retorne o texto que tiver maior tamanho.
- Teste a função com as seguintes situações:
  - `maior_str('python', 'fia')` **`-> python`**
  - `maior_str('maria', 'ana')` **`-> maria`**
  - `maior_str('python', 'codigo')` **`-> python e codigo tem o mesmo tamanho`**
- Não é necessário tratar outros tipos de dados (nesse momento).

Argumentos

</> Abra o arquivo **"aula3-parte1-exercicios-revisão-gabarito.ipynb"**

Conteúdo da Aula

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

Novos Conceitos

- Iremos aprender dois novos conceitos que serão utilizados no futuro:
  - Compreensão de Listas
  - Funções Anônimas

Compreensão de Lista

- Compreensão de Lista é uma ferramenta para transformar uma lista em outra lista.
- Durante a transformação, cada elemento pode ser condicionado e/ou transformado.
- A Compreensão de lista é mais compacta e rápida do que o laço de repetição (for) para construir uma lista.

Compreensão de Lista

- Considere o seguinte exemplo:

```
>>> lista = [1, 2, 3, 4, 5]
```

- Como podemos adicionar o valor 10 para cada item da minha lista?

```
>>> for i in range(len(lista)):
        lista[i] += 10
>>> print(lista)
[11, 12, 13, 14, 15]
```

Compreensão de Lista

- Isso funciona, porém pode não ser a abordagem ótima de "melhores práticas" no Python.
- Hoje, o conceito de Compreensão de lista faz com que os padrões de codificação sejam obsoletos.
- Assim, podemos substituir o laço com um única expressão que produz o mesmo resultado.

```
>>> lista = [x + 10 for x in lista]
>>> print(lista)
[21, 22, 23, 24, 25]
```

Compreensão de Lista

- Compreensão de lista é um conceito mais conciso para escrever e como esse padrão de código de construir as listas é tão comum no Python, elas se revelam muito úteis em muitos contextos.
- Além disso, dependendo da versão do Python e do código, a compreensão de lista **pode executar muito mais rápido do que as instruções manuais do laço, pois suas iterações são executadas na velocidade da linguagem C.**
- Principalmente para conjuntos de dados maiores, muitas vezes há uma grande vantagem de desempenho ao usar essa expressão.

Compreensão de Lista

- A sintaxe básica é:

```
[exprMap for elemento in listaOrigim if exprDeFiltragem]
```

  - ExpMap – Expressão de mapeamento
  - listaOrigim – Lista original
  - expDeFiltragem – no caso de utilizar alguma pequena expressão de controle

</> Abra o arquivo **"aula3-parte2-novos-conceitos.ipynb"**

Compreensão de Lista

- Considere a seguinte lista:

```
>>> nomes = ['maria', 'pedro', 'marcos', 'paulo', 'joao’]
```

- Como podemos filtrar apenas os nomes que começam com m utilizando o conceito de compreensão de lista ?

```
>>> nomes_filtrados = [x for x in nomes if x.startswith('m')]
>>> print(nomes_filtrados)
['maria', 'marcos']
```

Compreensão de Lista

- Também podemos utilizar laço **aninhado**, considere o seguinte código:

```
>>> res = []
>>> texto1 = 'abc'
>>> texto2 = '12'
>>> for i in texto1:
        for j in texto2:
            res.append(i + j) >>> print(res)
>>> print(res)
['a1', 'a2', 'b1', 'b2', 'c1', 'c2']
```

Compreensão de Lista

- Utilizando compreensão de listas:

```
>>> res = [i + j for i in texto1 for j in texto2]

>>> print(res)

['a1', 'a2', 'b1', 'b2', 'c1', 'c2']
```

Funções Anônimas

Funções Anônimas

- Já programamos com os paradigmas Orientado a Objeto, bem como o Estruturado.
- O Python também possibilita trabalhar com o paradigma de **programação funcional**.
- Vantagens:
  - Redução do código-fonte
  - Maior velocidade
  - Em alguns casos facilita as implementações.
- Desvantagens:
  - Maior possibilidade de gerar códigos obscuros
  - Não é um paradigma muito difundido.

Funções Anônimas

- As expressões lambdas são funções que não precisam ser nomeadas, chamadas de **funções anônimas**.
- As expressões lambdas são úteis quando usadas com as funções `filter`, `map` e `reduce` do Python.

- Exemplo genérico:

```
lambda arg1, arg2: expressão
```

Funções Anônimas

- **`Filter`** permite realizar a filtragem de elementos de uma estrutura não-escalar.

```
>>> nums = [2, 6, 8, 12]
>>> res = filter(lambda x : x % 3 == 0, nums)
>>> print(list(res))
[6, 12]
```

</> Abra o arquivo **"aula3-parte2-novos-conceitos.ipynb"**

Funções Anônimas

- **Map** permite realizar o mapeamento dos elementos de um estrutura não-escalar para uma nova estrutura não-escalar aplicando uma função.

```
>>> nums = [2, 6, 8, 12]
>>> res = map(lambda x : x * 2, nums)
>>> print(list(res))
[4, 12, 16, 24]
```

</> Abra o arquivo **"aula3-parte2-novos-conceitos.ipynb"**

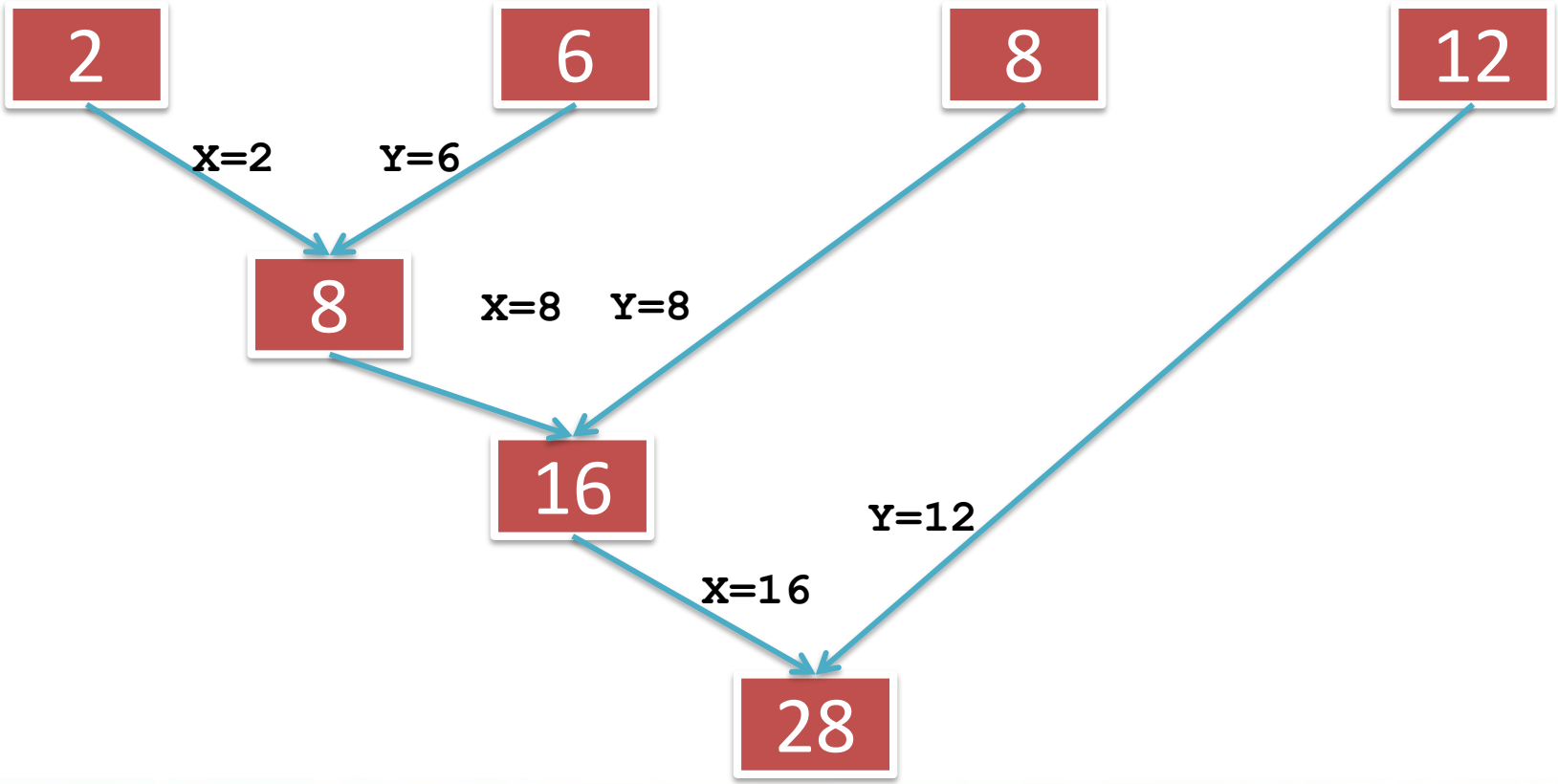
Funções Anônimas

- **Reduce** permite aplicar uma função em uma estrutura não-escalar para retornar uma valor único no final.

```
>>> import functools
>>> nums = [2, 6, 8, 12]
>>> res = functools.reduce(lambda x,y : x + y, nums)
>>> print(res)
28
```

</> Abra o arquivo **"aula3-parte2-novos-conceitos.ipynb"**

Funções Anônimas



Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- Novos conceitos
- **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

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.

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

http://www.statsmodels.org

- É um biblioteca Python que fornece classes e funções para estimativa de diversas funções estatísticas.
  - 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

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.

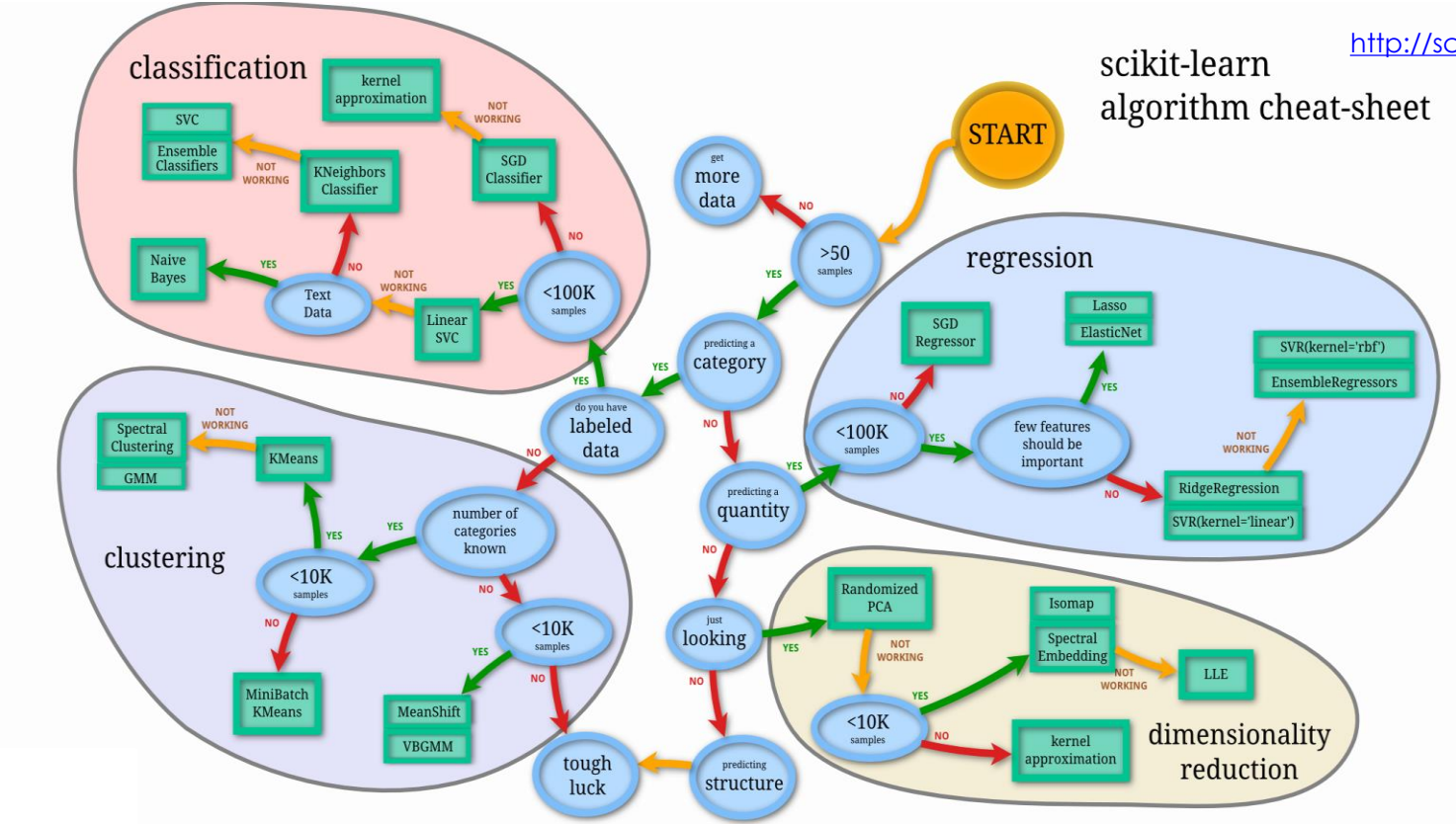
Scikit-learn

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/



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

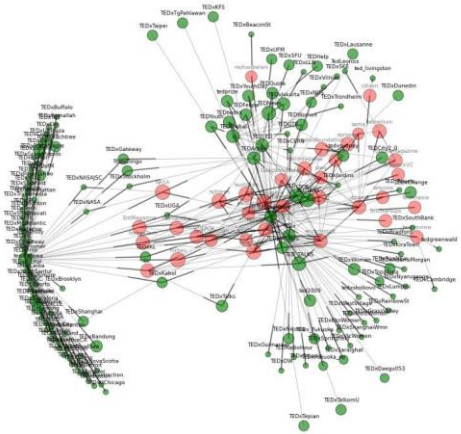
https://bokeh.pydata.org/

- 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

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
- Novos conceitos
- 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 contínuas - 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 em 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, JSON entre outros.
  - 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 matplotlib e scikit-lean.

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

- 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 linguagens de programação como R.

Pandas

- Carregando nosso primeiro conjunto de dados.
- Utilizaremos o conjunto de dados Gapminder (**`dados.tsv`**) que tem as seguintes características:

Variável
pais
continente
ano
expectativa vida
populacao
pib

</> Abra o arquivo **"aula03-parte3-intro.ipynb"**

Fonte: https://github.com/jennybc/gapminder

Pandas

- Para abrir um arquivo separado por tabs no Pandas, temos que primeiro importar a biblioteca dentro do Notebook.

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

- Depois utilizaremos a função read_csv para carregar um arquivo de dados separados por algum caractere.

```
>>> df = pd.read_csv('dados.tsv', sep='\t')
```

```
>>> print(type(df))
```

Pandas

- Também podemos obter o número de linhas e de colunas.

```
>>> df.shape
```

- Podemos utilizar o atributo columns para recuperar o nome de todas as colunas.

```
>>> print(df.columns)
```

- Podemos visualizar os tipos (dtypes) de cada coluna.

```
>>> print(df.dtypes)
```

- Por fim, podemos obter mais informações sobre nossos dados.

```
>>> print(df.info())
```

Pandas

- Como vimos, quando carregamos o arquivo estruturado, Pandas irá criar uma estrutura de dados do tipo **DataFrame**.
- Esse é o tipo comumente utilizado e para acessar todo o conteúdo de um determina coluna, utilizamos a mesma ideia de acessar elementos dentro de um dicionário Python.
- O comando `df['continente']` irá retornar todas as linhas da coluna em questão.
- Porém para acessar uma ou mais linhas específicas, podemos utilizar o nome (rótulo) ou índices.

Pandas

- Os métodos disponíveis que permitem o acesso as linhas são:

Método para obter subconjunto de linhas	Descrição
`.loc`	Subconjunto baseado no rótulo do índice (nome de linha)
`.iloc`	Subconjunto baseado no índice (número de linha)

</> Abra o arquivo **"aula03-parte3-intro.ipynb"**

Pandas

- Como vimos, quando criamos um subconjunto com apenas uma linha, o Pandas estrutura os dados no tipo **Series**.
- O Pandas permite acessar os dados via índice explicito (nome) e índice implícito (número).
- Em nosso conjunto de dados, tanto o índice explicito (nome) e o índice implícito (número) são os mesmos, mas temos que ter atenção pois eles podem ser diferentes.

Pandas

- Considere o seguinte exemplo:

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

</> Abra o arquivo **"aula03-parte3-intro.ipynb"**

Pandas

- 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
```

Pandas

- O atributo `iloc` 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
```

Pandas

- Também podemos utilizar o `loc` e o `iloc` para obter subconjuntos de colunas, linhas ou ambos.
- A sintaxe geral do `loc` e do `iloc` para esse cenário são:

</> Abra o arquivo **"aula03-parte3-intro.ipynb"**

```
df.loc[[linhas], [colunas]]

df.iloc[[linhas], [colunas]]
```

Pandas - Series

- Agora que tivemos uma introdução ao Pandas, vamos entender como é possível criar manualmente as duas estruturas.
  - Series e DataFrame
- Para fazer uso dessas estruturas, primeiro precisamos importar a biblioteca.
  - `>>> import pandas as pd`

</> Abra o arquivo **"aula03-parte4-series.ipynb"**

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, atribuições, aplicar funções aritméticas e estatísticas, entre tantos outros.

</> Abra o arquivo **"aula03-parte4-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 e linhas 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`
  - Dados estruturados
  - Series
  - Outra estrutura DataFrame

</> Abra o arquivo **"aula03-parte5-dataframe.ipynb"**

Pandas – DataFrame

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

</> Abra o arquivo **"aula03-parte5-dataframe.ipynb"**

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- Novos conceitos
- 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 **"aula03-parte6-exemplo-pratico.ipynb"**

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- Novos conceitos
- 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** - Selecione os itens que tenham população maior que 1 milhão e área menor que 500 km2.

- **Exercício 4** - Selecione os itens que tenham população maior que 5 milhões ou área maior que 5000 km2.

Argumentos

</>

Abra o arquivo **"aula3-parte6-exercicios-gabarito.ipynb"**

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

- **Use a Cabeça! Python** – Paul Barry - Rio de Janeiro, RJ: Alta Books, 2012.

- **Use a Cabeça! Programação** – Paul Barry & David Griffiths – Rio de Janeiro RJ: Alta Books, 2010.

- **Aprendendo Python: Programação orientada a objetos** – Mark Lutz & David Ascher – Porto Alegre: Bookman, 2007

Referências Bibliográficas

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

- **Python for Data Analysis** – Wes McKinney – USA: O'Reilly, 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