# Introdução à linguagem Python

Thiago Martins

# As bibliotecas SciPY

- Um projeto *open source* de extensões Python para computação científica
- Apresenta-se como uma alternativa significativa a pacotes tradicionais de computação científica como o Matlab.
- Numpy
- Pandas
- Matplotlib

# Pandas

- PANel DAta (s?).
- Python Data Analysis Library (???)

https://pandas.pydata.org/

# Pandas

- Biblioteca para processamento de dados em forma *tabular* (tabelas).
- Implementa uma combinação de operações de planilhas eletrônicas (aplica expressões a dados), banco de dados relacional (*merge, join*) e software de estatística.

# Dataframe e Séries

- Objeto central: `Dataframe`
  - Representa dados de forma tabular, organizados em duas dimensões, por coluna e linha e *indexados*.
  - Uma coluna de um `Dataframe` é uma `Serie`. Os dados dentro de uma coluna são de um tipo *homogêneo.*
  - Mas tabelas podem agregar colunas (Séries) com tipos distintos.

# Series

- Representa um agregado unidimensional *homogêneo* e *indexado*.
  - Índices: Uma sequência de valores (idealmente únicos) empregada para referenciar uma entrada na sequência. O conceito se aplica tanto a índices de séries quando de DataFrames.
    - Algumas operações *exigem* índices únicos e lançarão uma exceção caso esta condição não seja atendida.

# Series

- Construtor básico:

  `pd.Series(data, index=idx)`

  - `data` é qualquer dado enumerável de tamanho conhecido.
  - `idx` (opcional) é o índice.

- Exemplo Simples:

  `pd.Series(range(10))`

# Series

- Construtor básico:

  `pd.Series(data, index=idx)`

  - `data` é qualquer dado enumerável de tamanho conhecido.
  - `idx` (opcional) é o índice.

- Exemplo Simples:

  `pd.Series(range(10))`

# Series

- Construtor básico:

  `pd.Series(data, index=idx)`

  - `data` é qualquer dado enumerável de tamanho conhecido.
  - `idx` (opcional) é o índice.

- Exemplo Simples:

  `pd.Series(range(10))`

# Series

- Exemplos:

```
pd.Series({'a': 1, 'b': 2})
```

  - Quando criada a partir de um dicionário, os valores do índice são obtidos automaticamente das chaves.

```
pd.Series([1,2,3],index=[3,2,1])
```

  - Índice criado explicitamente.

# Series

- Exemplos:

```
pd.Series({'a': 1, 'b': 2, 'd': 3}, index=['a', 'b', 'c'])
```

  - Se índicies são passados explicitamente, somente os dados com chaves correspondentes são usados.
  - O valor nan (Not A Number) é usado por Pandas para representar dados faltantes.

```
pd.Series(1, index=['a', 'b', 'c'])
```

- Quando um escalar é passado, é replicado para todos os índices.

# Series - Acessando

- Uma Serie funciona como um dicionário, acessado pelos índices.
- Séries são *mutáveis*.

```
a = pd.Series(range(3) index=['a', 'b', 'c'])
```

```
a['a']=10
```

- Para além dos seus índices, uma série tem também uma ordenação implícita baseada em inteiros. É possível fazer *slices* empregando esta ordenação:

```
a[0:1]
```

# Series - Operações

- É possível fazer operações entre series como se estas fossem objetos do tipo ndarray.

```
a = pd.Series(np.arange(0,2*np.pi, 0.01))

np.sin(a)
```

- Quando uma operação é feita entre duas séries, estas são alinhadas pelos seus índices.

```
a = pd.Series(range(3) index=['a', 'b', 'c'])

b = pd.Series(range(3) index=['b', 'c', 'd'])

a+b
```

  - Uma operação entre duas séries com conjuntos de índices desiguais gera uma série cujos índices são a *união* dos índices dos operandos.
  - Novamente, dados faltantes são substituídos por nan.

# Dataframes – Construíndo

- Há uma grande variedade de maneiras de se construir um Dataframe.
- A partir de Sequências de sequências:

```
pd.DataFrame([[1,2],[3,4]])
```

Cada sequência é uma nova *linha*.

Além de um índice, um DataFrame também tem rótulos em cada coluna. Assim como em índices na construção de Sequências, quando estes são omitidos, valores baseados em zero são criados automaticamente.

É possível passar explicitamente o rótulo das colunas.

```
pd.DataFrame([[1,2],[3,4]], columns=["a", "b"])
```

# Dataframes – Construíndo

- Combinando dicionários de sequências:

```
pd.DataFrame({"C1": range(3), "C2": ['a', 'b', 'c']})
```

Cada dicionário representa uma coluna

- Combinando dicionários de *dicionários(!)*:

```
pd.DataFrame({"A":{1: 'a', 2: 'b'}, "B":{2: 'a', 3: 'b'}})
```

Novamente, os índices finais são a *união* dos índices de cada dicionário.

- Combinando sequências de dicionários:

```
pd.DataFrame([{1: 'a', 2: 'b'}, {2: 'a', 3: 'b'}])
```

- Note como neste caso cada sequência é uma *linha*.

# Dataframes – Construíndo

- Combinando dicionários de sequências:

```
pd.DataFrame({"C1": range(3), "C2": ['a', 'b', 'c']})
```

Cada dicionário representa uma coluna

- Combinando dicionários de *dicionários(!)*:

```
pd.DataFrame({"A":{1: 'a', 2: 'b'}, "B":{2: 'a', 3: 'b'}})
```

Novamente, os índices finais são a *união* dos índices de cada dicionário.

- Combinando sequências de dicionários:

```
pd.DataFrame([{1: 'a', 2: 'b'}, {2: 'a', 3: 'b'}])
```

- Note como neste caso cada sequência é uma *linha*.

# Dataframes – Construíndo

- Series comportam-se como dicionários:

```
a=pd.Series([1,2,3],index=['a', 'b', 'c'], name="linha1")
b=pd.Series([2,3,4],index=['b', 'c', 'd'], name="linha2")
pd.DataFrame((a,b))
```

Os *nomes* de cada série viram índices.

- É possível *transpor* o Dataframe (como em uma matriz).

```
a=pd.Series([1,2,3],index=['a', 'b', 'c'], name="col1")
b=pd.Series([2,3,4],index=['b', 'c', 'd'], name="col2")
pd.DataFrame((a,b)).T
```

# Dataframes – Acessando

- O índice externo de um dicionário referencia uma *coluna* (é uma convenção *distinta* de matrizes!):

```
a=pd.DataFrame({“A”:[1,2,3], “B”:[4,5,6]})
a[“A”]
```

Um Dataframe é *mutável*. É possível modificar colunas, inserir, remover...

- Ao adicionar uma nova sequência a um Dataframe, índices são alinhados. Os índices finais são (novamente!) uma *união* dos índices da sequência adicionada e dos índices da sequência adicionada.

# Dataframes – Acessando

- O índice externo de um dicionário referencia uma *coluna* (é uma convenção *distinta* de matrizes!):

```
a=pd.DataFrame({“A”:[1,2,3], “B”:[4,5,6]})
a[“A”]
```

Um Dataframe é *mutável*. É possível modificar colunas, inserir, remover...

- Ao adicionar uma nova sequência a um Dataframe, índices são alinhados. Os índices finais são (novamente!) uma *união* dos índices da sequência adicionada e dos índices da sequência adicionada.

# Dataframes – Acessando

- Método `assign` permite criar um novo dataset com novas colunas...
  - Com valores pré-computados...
  - Mas com funções também

```
a=pd.DataFrame({“theta”:np.arange(0,2*np.pi,0.01)})
b=a.assign(sin = lambda x: np.sin(x))
```

# Dataframes – Acessando

- O atributo `loc` permite endereçar primariamente por linhas.

```
a=pd.DataFrame([[1,2],[3,4]])
a.loc[0]
```

- Também é possível endereçar colunas:

```
a.loc[0,0]
```

# Dataframes – Acessando

O atributo `iloc` permite endereçar linhas *numericamente* (mesmo quando o índice não é numérico):

```
a=pd.DataFrame({“C1”:{‘A’:1,‘B’:2}})
a.iloc[1,0]
```

# Dataframes – Acessando

- É possível acessar (e modificar!) também os rótulos das colunas e o índice, via atributos `columns` e `index`.

```
a=pd.DataFrame([[1,2],[3,4]])
a.columns=[“c1”, “c2”]
a.index=[“l1”, “l2”]
a.loc[“l1”, “c2”]
```

# Dataframes  e Sequências – Máscaras

- Pode-se selecionar um subconjunto de elementos por meio de uma máscara booleana.

```
pd.Series(range(4))[[False, True, False, True]]
```

- Os valores selecionados podem ser *modificados*:

```
a = pd.Series(range(4))
a[[False, True, False, True]]=0
```

- Máscaras são geralmente obtidas como resultado de operações booleanas sobre Dataframes e Series (mais adiante).

# Dataframes  e Sequências – Operações

- Como em numpy, operações aritméticas são propagadas por todos os dados.

```
pd.Series(range(4))+pd.Series([1,2,1,2])
```

- Valores são alinhados por *índices* e (no caso de DataFrames) por *colunas*:

```
pd.DataFrame([[1,2],[3,4]])+pd.DataFrame(1, index=[1,2], columns=[1,2])
```

# Dataframes  e Sequências – Operações

- Escalares são propagados para séries e dataframes.

```
pd.Series(range(4))+1
```

- Séries são propagadas para DataFrames coluna-a-coluna, alinhando-se os *índices* da série com as *colunas* do DataFrame:

```
pd.DataFrame([[1,2],[3,4]])+pd.range(1, index=[1,2])
```

# Dataframes  e Sequências – Operações

- Métodos explícitos conferem controle fino a estas operações (`add()`, `sub()`, `mult()`, div()...)
- Opção `axis`: Escolhe qual o eixo do DataFrame será alinhado com o índice da sequência (0=índice, 1=colunas)
- Opção `fill_value`: Escolhe o valor que deve ser usado no caso de operandos faltantes (devido a desalinhamento de dados).

# Dataframes  e Sequências – Operações

- Método combine: Uma versão genérica de operação entre dois conjuntos de dados.

`combine(other, func, fill_value=(opcional), overwrite=(opcional)`:

- `combine_first(other)`:  Adiciona a self todos os dados em other. Os dados já existentes em self são mantidos.

```
s1=pd.Series({0:1,1:1,2:0)
s2=pd.Series({1:1,2:1,3:200})
s1.combine(s2,max,fill_value=100)
```

# Dataframes  e Sequências – Operações

- Agregar:
- max(): Máximo
- min(): Mínimo
- average(): Média
- std(): Variância populacional
- size(): Tamanho
- count(): Contagem (não NaN)
- first(): Primeiro
- last(): Último

# Dataframes  e Sequências – Operações

- Operações de comparação funcionam como em numpy, produzindo um Dataset/Serie de valores booleanos.
- Operações de *redução* booleana que convertem um dataset inteiro a um único valor:
    - `all()`: Verdadeiro se *todos* os valores são verdadeiros
    - `any()`: Verdadeiro se *algum* valor é verdadeiro.
- Exemplo: verdadeiro se todos os elementos de a são positivos: `(a > 0).all()`

# Dataframes  e Sequências – Operações

- Lembrete: um DataFrame/Serie booleana pode ser usado como máscara para seleção!
- Exemplo. Atribui zero a todos os elementos de a negativos:

```
a[a<0] = 0
```

# Dataframes  e Sequências – apply

- Em sequências: aplica uma função a cada um dos componentes.
- Em DataFrames: Aplica uma função a cada coluna/linha.

# Combinando Dataframes - concat

```
pd.concat(objs, axis=0, join='outer',
ignore_index=False, keys=None,
verify_integrity=False)
```

- Produz uma nova série concatenando os objetos em `obj`. Esta operação pode produzir índices repetidos.

```
pd.concat([pd.Series(['a','b']),pd.Series(['c','d'])])
```

- O parâmetro `ignore_index` , se definido como True, constrói um índice novo.

```
pd.concat([pd.Series(['a','b']),pd.Series(['c','d'])],
ignore_index=True)
```

# Combinando Dataframes - concat

```
pd.concat(objs, axis=0, join='outer',
ignore_index=False, keys=None,
verify_integrity=False)
```

- Produz uma nova série concatenando os objetos em obj. Esta operação pode produzir índices repetidos.

```
pd.concat([pd.Series(['a','b']),pd.Series(['c','d'])])
```

- As chaves em keys criam um novo índice hierárquico.

```
pd.concat([pd.Series(['a','b']),pd.Series(['c','d'])],
keys=['serie1', 'serie2'])
```

# Combinando Dataframes - concat

```
pd.concat(objs, axis=0, join='outer',
ignore_index=False, keys=None,
verify_integrity=False)
```

- Em dataframes, axis define o eixo: 0 (padrão) concatena adicionando linhas. 1 Concatena adicionando colunas

```
d1=pd.DataFrame([['a',1],['b',2]],columns=['l','n'])
d2=pd.DataFrame([['c',3,'uva'],['d', 4,'pera']],
    columns=['l','n','f'])
pd.concat([df1, df3])
pd.concat([df1, df3], axis=1)
```

# Combinando Dataframes - concat

```
pd.concat(objs, axis=0, join='outer',
ignore_index=False, keys=None,
verify_integrity=False)
```

- Em dataframes, o parâmetro `join` especifica como tratar os demais eixos (normalmente as colunas). ‘outer’ gera uma *união* dos valores dos outros eixos, ‘inner’ gera uma intersecção.

```
d1=pd.DataFrame([['a',1],['b',2]],columns=['l','n'])
d2=pd.DataFrame([['c',3,'uva'],['d', 4,'pera']],
    columns=['l','n','f'])
pd.concat([df1, df3])
pd.concat([df1, df3], join=’inner’)
```

# Combinando Dataframes - concat

```
pd.concat(objs, axis=0, join='outer',
ignore_index=False, keys=None,
verify_integrity=False)
```

- `verify_integrity` Garante que não há índices repetidos.

```
d1=pd.DataFrame([['a',1],['b',2]],columns=['l','n'])
d2=pd.DataFrame([['c',3,'uva'],['d', 4,'pera']],
    columns=['l','n','f'])
pd.concat([d1,d2], verify_integrity=True))
d3=pd.DataFrame({'l':{2:'c',3:'d'},'n':{2:3,3:4}})
pd.concat([d1,d3], verify_integrity=True))
```

# Combinando Dataframes - merge

- Similar a um *join* de uma base de dados relacional.

```
pd.merge(left, right, how='inner', on=None, left_on=None,
  right_on=None, left_index=False, right_index=False,
  sort=True,suffixes=('_x', '_y'))
```

- Combina dois DataFrames quando valores especificados pela coluna on são iguais.
- `left_index, right_index`: Especificam que os índices em left/right são usados como as chaves.
- how: inner: Intersecção das chaves. left/right: *Todas* as chaves de left/right e as equivalentes na outra. outer: união de todas as chaves.

# Agrupando dados: groupby

- Cria um objeto de agrupamento de dados com base em algum critério (geralmente, o nome de uma coluna).
- O resultado da operação são DataFrames que podem ser consultados individualmente.
- O uso mais comum é *agregar* o resultado de acordo com alguma função (np.sum, por exemplo, calcula o total).