FUNDAMENTOS E TÉCNICAS EM CIÊNCIAS DE DADOS

PROF. JOSENALDE OLIVEIRA

josenalde.oliveira@ufrn.br
https://github.com/josenalde/datascience

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

PANDAS #1 – I/O DATAFRAMES

- Coletando dados (**importando** ou **CRIANDO para validação...**)

```
import pandas as pd
df = pd.read_csv(....)
       .read_excel
       .read_html
       .read_json
       .json_normalize
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Observar número de parâmetros
Prática comum – consulta da referência!!

Por exemplo, no read_csv, são parâmetros comuns:
- sep=';', ',' etc.
- encoding='utf-8'

https://docs.python.org/3/library/codecs.html#standard-encodings

- Gravando (exportando) dados (normalmente após manipulação)

```
import pandas as pd
df = pd.read_csv(....)
...
df.to_csv(...)
  .to_excel
  .to_json
  .to_dict
  ...
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

PANDAS #1 – I/O DATAFRAMES

- Interoperabilidade com tipos Python

Dicionário (dict):

```
data = {'year': [
          2010, 2011, 2012,
          2010, 2011, 2012,
          2010, 2011, 2012
      ],
      'team': [
          'FCBarcelona', 'FCBarcelona',
          'FCBarcelona', 'RMadrid',
          'RMadrid', 'RMadrid',
          'ValenciaCF', 'ValenciaCF',
          'ValenciaCF'
      ],
      'wins':   [30, 28, 32, 29, 32, 26, 21, 17, 19],
      'draws':  [6, 7, 4, 5, 4, 7, 8, 10, 8],
      'losses': [2, 3, 2, 4, 2, 5, 9, 11, 11]
      }
football = pd.DataFrame(data, columns = [
      'year', 'team', 'wins', 'draws', 'losses'
      ]
)
```

Índices (index)

	year	team	wins	draws	losses
0	2010	FCBarcelona	30	6	2
1	2011	FCBarcelona	28	7	3
2	2012	FCBarcelona	32	4	2
3	2010	RMadrid	29	5	4
4	2011	RMadrid	32	4	2
5	2012	RMadrid	26	7	5
6	2010	ValenciaCF	21	8	9
7	2011	ValenciaCF	17	10	11
8	2012	ValenciaCF	19	8	11

data=data

Neste caso, embora especifique, as chaves já são as colunas

PANDAS #1 – BÁSICO

- Interoperabilidade com tipos Python

```
#Array NumPy:

import numpy as np
np.random.seed(5) # se desejar reprodutibilidade
# cria array aleatório 10x3 com inteiros entre 0 e 9, inclusive
data = np.random.randint(0,10,size=(10,3))
cols = list('ABC') #cria lista = [‘A’, ‘B’, ‘C’]
df = pd.DataFrame(data=data, columns=cols)
```

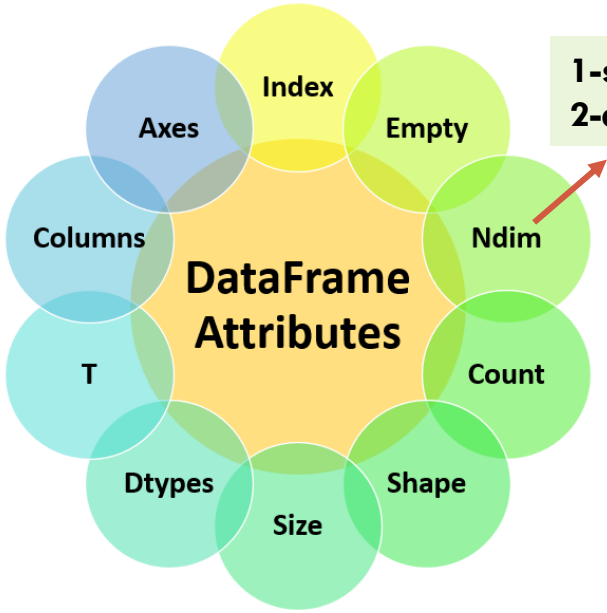
	A	B	C
0	3	6	6
1	0	9	8
2	4	7	0
3	0	7	1
4	5	7	0
5	1	4	6
6	2	9	9
7	9	9	1
8	2	7	0
9	5	0	0

```
1 df.describe()
```

	A	B	C
count	10.000000	10.000000	10.000000
mean	3.100000	6.500000	3.100000
std	2.766867	2.758824	3.695342
min	0.000000	0.000000	0.000000
25%	1.250000	6.250000	0.000000
50%	2.500000	7.000000	1.000000
75%	4.750000	8.500000	6.000000
max	9.000000	9.000000	9.000000

Uma vez disponível o DATASET, funções comuns:
head(), tail(), info(), describe()
Atributos:
shape, dtypes, columns, axes,
size...

Tupla (lin, col)



1-series
2-df

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       10 non-null     int32
 1   B       10 non-null     int32
 2   C       10 non-null     int32
dtypes: int32(3)
memory usage: 248.0 bytes
```

PANDAS #1 – DATAFRAMES X SERIES

Jupyter format

	YEARMODA	TEMP	MAX	MIN
0	20160601	65.5	73.6	54.7
1	20160602	65.8	80.8	55.0
2	20160603	68.4	77.9	55.6
3	20160604	57.5	70.9	47.3
4	20160605	51.4	58.3	43.2
5	20160606	52.2	59.7	42.8
6	20160607	56.9	65.1	45.9
7	20160608	54.2	60.4	47.5
8	20160609	49.4	54.1	45.7
9	20160610	49.5	55.9	43.0

Standard Python format

```
   YEARMODA  TEMP   MAX   MIN
0  20160601  65.5  73.6  54.7
1  20160602  65.8  80.8  55.0
2  20160603  68.4  77.9  55.6
3  20160604  57.5  70.9  47.3
4  20160605  51.4  58.3  43.2
5  20160606  52.2  59.7  42.8
6  20160607  56.9  65.1  45.9
7  20160608  54.2  60.4  47.5
8  20160609  49.4  54.1  45.7
9  20160610  49.5  55.9  43.0
```

Column labels, Data, Index, Pandas Series

Pandas DataFrame

`pandas.core.frame.DataFrame`

Standard Python format

```
0    65.5
1    65.8
2    68.4
3    57.5
4    51.4
5    52.2
6    56.9
7    54.2
8    49.4
9    49.5
Name: TEMP, dtype: float64
```

Index, Data, Column label, Data type

Pandas Series

`pandas.core.series.Series`

Na série, as chaves do dict são índices

```
s1 = pd.Series({'a': 1, 'b': 2, 'c': 3})
s2 = df['TEMP'] ou df.TEMP
s1.count() ?
```

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

```
1  s1.index
```

```
Index(['a', 'b', 'c'], dtype='object')
```

```
pd.DataFrame({'float': [2.0],
              'int': [2],
              'datetime': [pd.Timestamp('20190210')],
              'string': ['f1']})
```

DataFrame	float	int	datetime	string
0	2.0	2	2019-02-10	'f1'

©w3resource.com

`df.head(1) ou df.tail(1)` # unica linha é Series

PANDAS #1 – TIPOS - NUMPY

Kind of Data	Data Type	Scalar	Array	String Aliases
tz-aware datetime	`DatetimeTZDtype`	`Timestamp`	`arrays.DatetimeArray`	`'datetime64[ns, <tz>]'`
Categorical	`CategoricalDtype`	(none)	`Categorical`	`'category'`
period (time spans)	`PeriodDtype`	`Period`	`arrays.PeriodArray`	`'period[<freq>]'`, `'Period[<freq>]'`
sparse	`SparseDtype`	(none)	`arrays.SparseArray`	`'Sparse'`, `'Sparse[int]'`, `'Sparse[float]'`
intervals	`IntervalDtype`	`Interval`	`arrays.IntervalArray`	`'interval'`, `'Interval'`, `'Interval[<numpy_dtype>]'`, `'Interval[datetime64[ns, <tz>]]'`, `'Interval[timedelta64[<freq>]]'`
nullable integer	`Int64Dtype`, ...	(none)	`arrays.IntegerArray`	`'Int8'`, `'Int16'`, `'Int32'`, `'Int64'`, `'UInt8'`, `'UInt16'`, `'UInt32'`, `'UInt64'`
Strings	`StringDtype`	`str`	`arrays.StringArray`	`'string'`
Boolean (with NA)	`BooleanDtype`	`bool`	`arrays.BooleanArray`	`'boolean'`

```python
dft = pd.DataFrame({ "A": np.random.rand(3),
                     "B": 1,
                     "C": "foo",
                     "D": pd.Timestamp("20010102"),
                     "E": pd.Series([1.0] * 3).astype("float32"),
                     "F": False,
                     "G": pd.Series([1] * 3, dtype="int8")
                   })
```

```
A          float64
B            int64
C           object
D    datetime64[ns]
E          float32
F             bool
G             int8
dtype: object
```

	A	B	C	D	E	F	G
0	0.204556	1	foo	2001-01-02	1.0	False	1
1	0.699844	1	foo	2001-01-02	1.0	False	1
2	0.779515	1	foo	2001-01-02	1.0	False	1

Tipos padrão numéricos: float64, int64
astype permite typecasting

PANDAS #1 – TIPOS - NUMPY

```
#dicionário - datas passadas como string dd-mm-aaaa
raw_data = {'nome': ['Pedro', 'Joaquim',
                     'Marcos', 'Mateus'],
            'age': [20, 19, 22, 21],
            'cor_favorita': ['azul', 'amarelo',
                             'cinza', 'verde'],
            'nota_final': [8.8, 9.2, 9.5, 7.0],
            'data_nasc': ['01-02-2000', '08-05-1997',
                          '04-28-1996', '12-16-1995']}
raw_data
df = pd.DataFrame(raw_data,
                  index = ['Pedro', 'Joaquim',
                           'Marcos', 'Mateus'])
# Cria coluna ano, usando o DatetimeIndex
df['ano'] = pd.DatetimeIndex(df['data_nasc']).year

# Cria a coluna mes, usando o DatetimeIndex
df['mes'] = pd.DatetimeIndex(df['data_nasc']).month
```

	nome	age	cor_favorita	nota_final	data_nasc	ano	mes
Pedro	Pedro	20	azul	8.8	01-02-2000	2000	1
Joaquim	Joaquim	19	amarelo	9.2	08-05-1997	1997	8
Marcos	Marcos	22	cinza	9.5	04-28-1996	1996	4
Mateus	Mateus	21	verde	7.0	12-16-1995	1995	12

```
= pd.DataFrame({ "A": np.random.rand(3),
                 "B": 1,
                 "C": "foo",
                 "D": pd.Timestamp("20010102"),
                 "E": pd.Series([1.0] * 3).astype("float32"),
                 "F": False,
                 "G": pd.Series([1] * 3, dtype="int8")
               })
```

```
float64
int64
object
datetime64[ns]
float32
bool
int8
ype: object
```

	A	B	C	D	E	F	G
0	0.204556	1	foo	2001-01-02	1.0	False	1
1	0.699844	1	foo	2001-01-02	1.0	False	1
2	0.779515	1	foo	2001-01-02	1.0	False	1

Tipos padrão numéricos: float64, int64
astype permite typecasting

OUTRA FORMA COM DATAS

```
import datetime as dt

df['ano'] = df['data_nasc'].dt.year
```

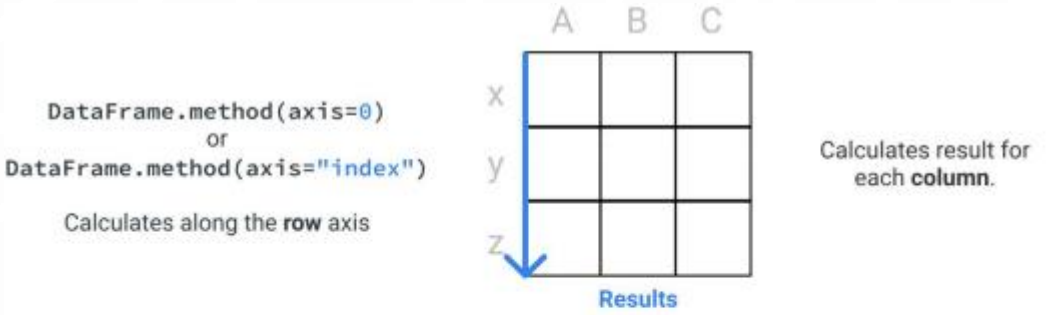
PANDAS #1 – SELECIONANDO

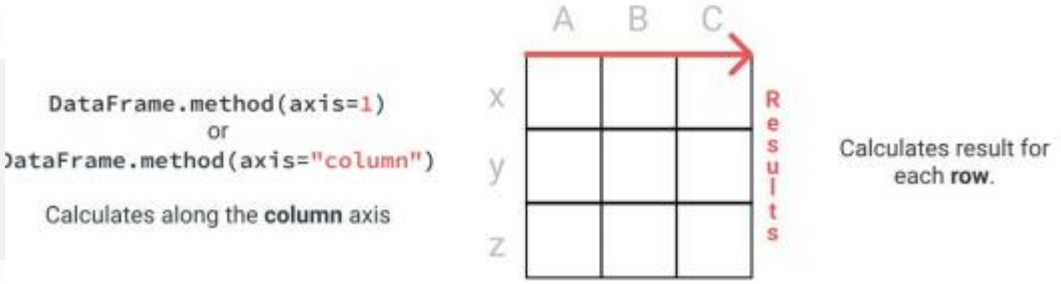
```
Pelo nome da coluna: df['wins'] # ou df.wins ou df.loc[:, 'wins']
```

	year	team	wins	draws	losses
0	2010	FCBarcelona	30	6	2
1	2011	FCBarcelona	28	7	3
2	2012	FCBarcelona	32	4	2
3	2010	RMadrid	29	5	4
4	2011	RMadrid	32	4	2
5	2012	RMadrid	26	7	5
6	2010	ValenciaCF	21	8	9
7	2011	ValenciaCF	17	10	11
8	2012	ValenciaCF	19	8	11

```
MÉTODO LOC: localizar conteúdo no DataFrame, passando linhas e colunas
            : - significa todas
```

Explorando por linha ou coluna - axis






```
Podemos acessar múltiplas colunas, estejam em sequência ou não
df.loc[:, ['wins', 'losses']] # ou df[['wins', 'losses']]

df.loc[:, 'wins':'losses']
```

Explorar o operador de faixa (range) **:**

Pode-se atribuir valores a uma coluna completa ou linhas específicas:

```
df['draws'] = 0 (altera toda a coluna draws)
df.loc[4,'wins'] = 35
```

```
df[['wins', 'draws']].median(axis=0) # axis='index'
wins 28.0 draws 7.0 dtype: float64
```

```
df[['wins', 'draws']].median(axis=1) #axis='column'
```

Para cada linha fará a média de wins com draws