Pandas

DEPARTAMENTO DE INFORMÁTICA PUC-RIO

PANDAS: Resumo1

(original: prof. Claudia Ferlin)

- ✓ O que é.
- ✓ Como usar.
- ✓ Estrutura: Series
  - Esquema simplificado de uma series
  - Criação
  - Acesso
  - Alteração
  - Inclusão
  - Exclusão
  - Ordenação
  - Categorização
  - Aplicação de função sobre valor
  - Visualização gráfica

DEPARTAMENTO DE INFORMÁTICA PUC-RIO

Pandas

Pandas é uma biblioteca de alto desempenho que fornece suporte para manipular dados estruturados bem como ferramentas para analisá-los.

Características principais:

- ✓ Indexação que permite fatiamento em diferentes perspectivas (*slice* e *dice*), agregações e seleção de subconjuntos de dados
- ✓ Conversão e mapeamento de dados de um estado "crú" para outro formato onde é possível utilizá-los em ferramentas de mais alto nível (Data Munging/Wrangling)

Principais Estruturas:

Series e DataFrame

Pandas: Como usá-lo

Não é um módulo built-in do Python mas com o Anaconda o Pandas é instalado automaticamente.

- ✓ **1º Passo)** Importar o(s) módulo(s):

  import pandas as pd

  Apelido do módulo

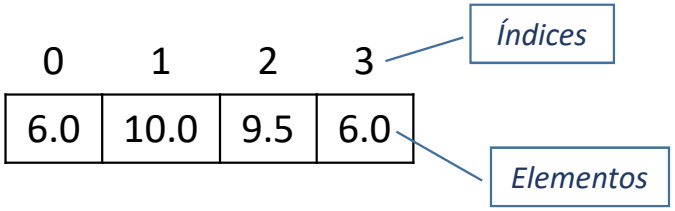
- ✓ **2º Passo)** Carregar o conjunto de dados no ambiente Python

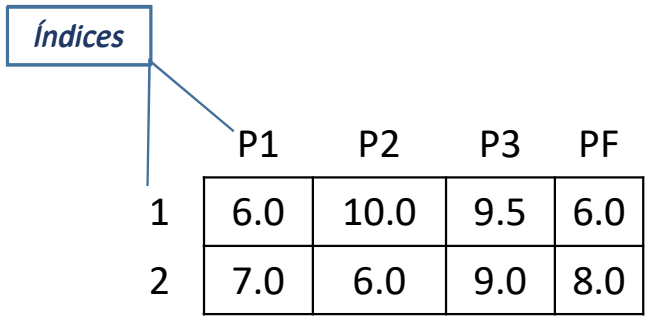
- ✓ **3º Passo)** Realizar operações e visualizações desejadas

Array

- Agregado de elementos de dados identificados por, pelo menos, um índice.
- Um elemento individual é identificado pela sua posição relativa ao primeiro elemento no agregado.
- A posição é determinada pelo índice que pode ser uma sequência de números inteiros ou de qualquer valor ordinal
- Também conhecida como arranjo.
  - **Vetor:** *array* unidimensional
  - **Matriz:** *array* bidimensional.

Exemplos:





Series do Pandas

Series: *array* unidimensional indexado que armazena valores de qualquer tipo.

Estrutura serial, similar a um vetor, lista, linha ou coluna de uma tabela, composta por:

- **valores**: uma sequência de int, string, float, list, dict, objetos Python, etc.
- **índices** (um por valor): uma sequência de números ou rótulos quaisquer (*labels)*

✓ Os índices não precisam ser exclusivos. Por padrão, variam de 0 a itens -1

Exemplo: Duas *Series* que armazenam os gastos com alimentação em cada dia da semana

Índice Padrão

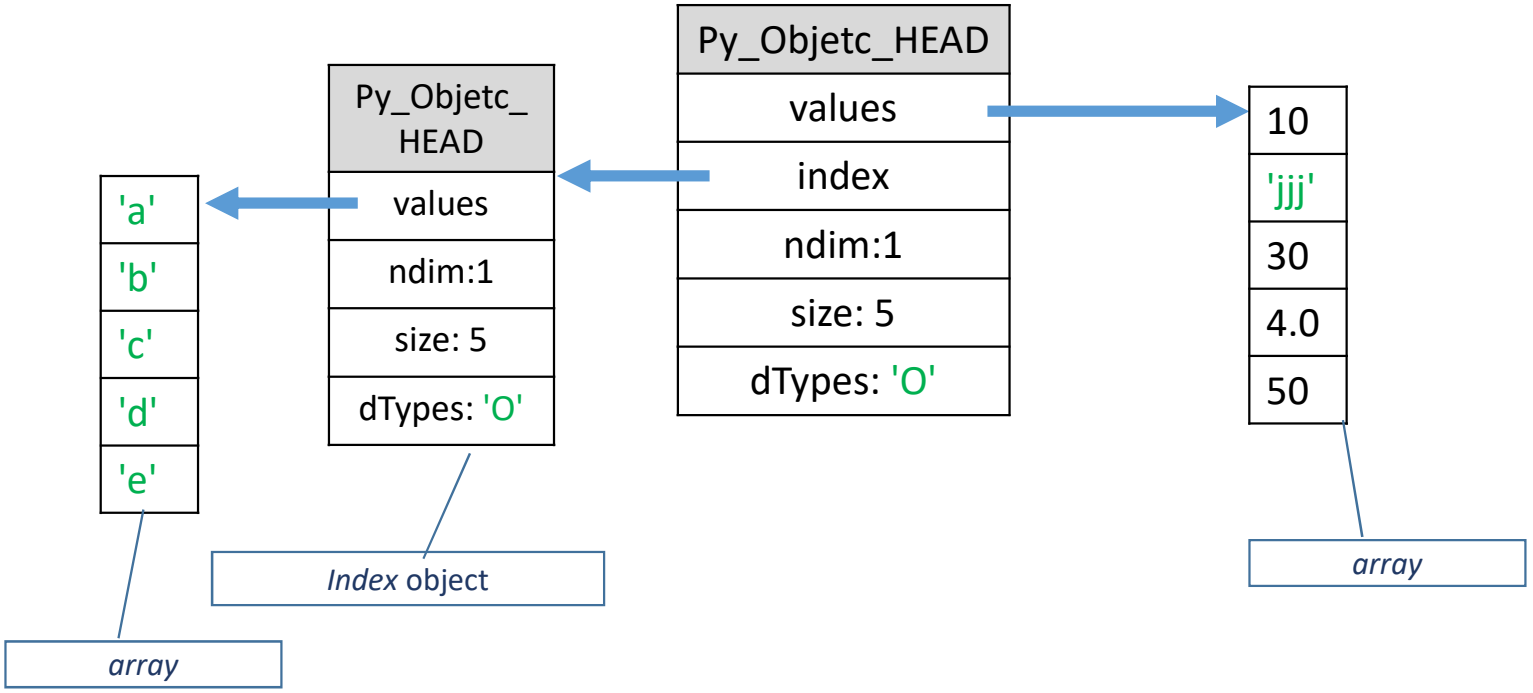
0	10.0
1	23.0
2	22.4
3	10.0
4	15.0
5	12.0
6	25.0

Índice Dado

Seg	10.0
Ter	23.0
Qua	22.4
Qui	10.0
Sex	15.0
Sab	12.0
Dom	25.0

Seg	10.0
Ter	23.0
Seg	22.4
Ter	10.0
Sex	15.0
Sex	12.0
Dom	25.0

Esquema simplificado do objeto Series



Criar/Acessar/Alterar/Incluir/Excluir

Construindo uma *Series*

sem especificação dos índices	`pd.Series(valores)`
com especificação dos índices	`pd.Series(valores , index = array unidimensional )`
a partir de um arquivo Excel	`pd.read_excel(caminho, sheet_name=0, usecols=None, index_col= squeeze=True, header=None, decimal=',').squeeze()` **caminho -** localização do arquivo: composto pelo caminho (absoluto/relativo) e nome **sheet_name= '....' –** nome da planilha (dafault 0) **usecols=** tupla com as colunas desejadas (default- None) **index_col =** *n* - em geral **0** . O número da coluna do arquivo (ou das selecionadas por usecols) a ser usada como *labels* do índice. None é o padrão usado quando o arquivo não possui tal coluna. ~~**squeeze = True** - se o arquivo tem apenas uma coluna, retorna uma Series~~ **squeeze agora é método!!! => pd.read.excel(...).squeeze('columns')** **header = None -** para arquivos que não possuem linha de cabeçalho ~~**decimal = ',' - quando o separador de casas decimais é a vírgula, Padrão: '.'~~

Atributos e Exibição

Tamanho	`series.size`
Valores	`series.values`
Labels dos Índices	`series.index`
Primeiros Elementos	`series.head(n)  default, n=5`
Últimos Elementos	`series.tail(n) default, n=5`

```
sF1_l:
0    9217
1    1118
2     665
3    3348
4    3599
```

```
sF1_d:
bmp    9217
gif    1118
jpg     665
png    3348
gif    3599
```

Operações Básicas

Acesso	*series*.loc[*índice*] *series*.loc[*lista de índices*]	Retorna o valor do elemento indexado por *índice* ou uma nova Series com os elementos da *lista de índices*. OBS: **.loc** não é método
	series.iloc[*posição*] *series*.iloc[*lista de posições*]	Retorna o valor do elemento indexado pela posição no Index ou uma nova Series com os elementos da *lista de posições de índices*.
Inclusão Alteração	*series*.loc[*índice*]= valor *series*.loc[lista de *índices*]= valor ou lista de valores	Altera o valor/valores do elemento(s) indexado(s) por *índice/lista de índices.* Se o índice não existe, é incluído.
Exclusão	*series*.drop(índice ou lista de índices)*	Retorna uma cópia da *series* sem os elementos da lista de índices. * com inplace=True, realiza a operação na Series, não cria uma cópia
Exclusão de valor NaN	*series*.dropna()*	Retorna uma cópia da *series* sem os elementos com valor NaN * com inplace=True, realiza a operação na Series, não cria uma cópia

Exemplo: Acesso de valores da *Series*

sGnum

```
0   10.0
1   23.0
2   22.4
3   10.0
4   15.0
5   12.0
6   25.0
```

sGdia

```
Seg   10
Ter   23
Qua   22.4
Qui   10
Sex   15
Sab   12
Dom   25
```

```
sGnum.loc[2]
22.4

sGnum.loc[0:2]
0    10.0
1    23.0
2    22.4
dtype: float64

sGnum.loc[[0,2]]
0    10.0
2    22.4
dtype: float64

sGnum.loc[99]
KeyError: 'the label [99] is not in
the [index
```

```
sGdia.iloc[2]
22.4

sGdia.iloc[0:2]
Seg    10.0
Ter    23.0
dtype: float64

sGnum.iloc[0:2]
0    10.0
1    23.0
dtype: float64

sGdia.iloc[9]
IndexError: single positional
indexer is out-of-bounds
```

```
sGdia.loc['Qua']
22.4

sGdia.loc['Seg':'Qua']
Seg    10.0
Ter    23.0
Qua    22.4
dtype: float64

sGdia.loc[['Seg','Qua']]
Seg    10.0
Qua    22.4
dtype: float64

sGdia.loc['oi']
KeyError: 'the label [oi] is
not in the [index]'
```

Exemplo: Alteração/Inclusão de valores na *Series*

```
>>>dInsc2 = {'33A':4,'33E':None,'33C':1}
>>>sInsc2=pd.Series(dInsc2)
>>>sInsc2
```

dInsc2:

33A 4
33C 1
33E NaN

```
>>> sInsc2.loc['33E']=18
>>>sInsc2
33A     4.0
33C     1.0
33E    18.0
dtype: float64

>>>sInsc2.iloc[1:]=[5,3]
>>>sInsc2
33A     4.0
33C     5.0
33E     3.0
dtype: float64
```

```
>>>sInsc2.iloc[1:]=2
>>>sInsc2
33A     4.0
33C     2.0
33E     2.0
dtype: float64

>>>sInsc2.loc['33B']=9
>>>sInsc2
33A     4.0
33C     2.0
33E     2.0
33B     9.0
dtype: float64
```

Exemplo: Descarte de elementos da Series

s:

```
33A   40.0
33B   15.0
33C   18.0
33E    NaN
33A   46.0
```

```
>>>s.drop(['33A','33E'])
33B    15.0
33C    18.0
dtype: float64

>>>s
33A    40.0
33B    15.0
33C    18.0
33E     NaN
33A    46.0
dtype: float64
```

```
>>>s.drop(['33A','33E'],inplace=True)
33B    15.0
33C    18.0
dtype: float64

>>>s
33B    15.0
33C    18.0
dtype: float64
```

Exemplo: Descarte de elementos com valor NaN

s:

```
33A   40.0
33B   15.0
33C   18.0
33E    NaN
33A   46.0
```

```
>>>s.dropna()
33A    40.0
33B    15.0
33C    18.0
33D    46.0
dtype: float64

>>>s
33A    40.0
33B    15.0
33C    18.0
33E     NaN
33A    46.0
dtype: float64
```

```
>>>s.dropna(inplace=True)
33A    40.0
33B    15.0
33C    18.0
33D    46.0
dtype: float64

>>>s
33A    40.0
33B    15.0
33C    18.0
33D    46.0
dtype: float64
```

DEPARTAMENTO DE INFORMÁTICA PUC-RIO

Métodos Úteis

sobre a Series

`series.unique()`	Retorna os valores exclusivos da Series
`series.nunique()`	Retornar o número de valores exclusivos na Series
`series.sort_values()` *	Retorna uma cópia da Series ordenada pelos valores *com **inplace=True**, realiza a operação na Series, **não** cria uma cópia, *com **ascending=False**, ordem não crescente
`series.sort_index()` *	Retorna uma cópia da Series ordenada pelos labels do index *com **inplace=True**, realiza a operação na Series, **não** cria uma cópia *com **ascending=False**, ordem não crescente
`series.reindex(labels)`	Retorna uma cópia da Series na ordem especificada pela sequência de labels recebida Não pode ser aplicado sobre índices com repetição

sobre o *Index*

`series.index.unique()`	Retorna os valores exclusivos do Index
`series.index.nunique()`	Retornar o número de valores exclusivos no Index
`series.index.get_loc(label)`	• Retorna: a posição do label no índice, qdo são exclusivos ou um array de booleanos: qdo há repetição

Categorizando os valores da Series

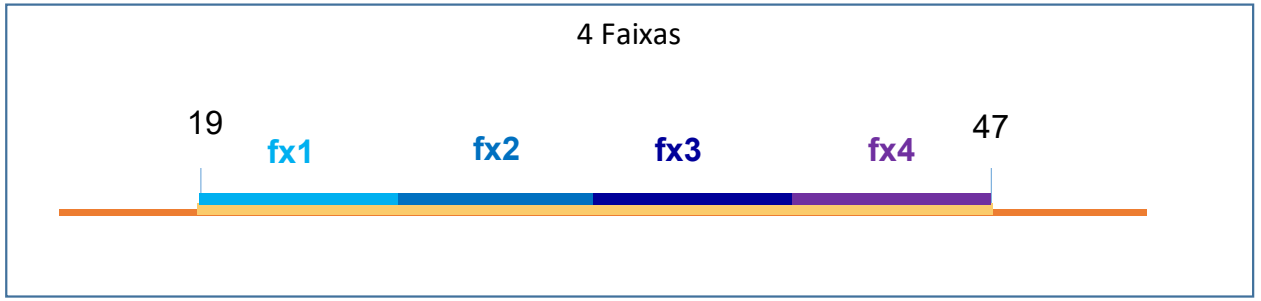
pandas.cut

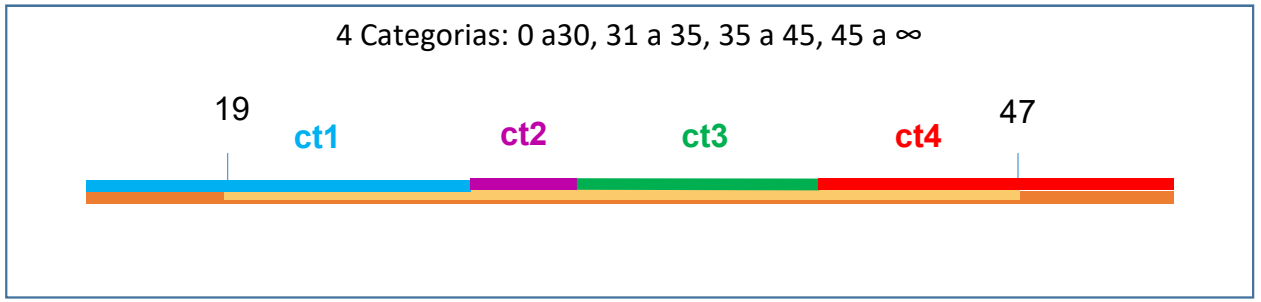
Útil para dados numéricos em escalas muito grandes ou muito granularizado:

a) organiza os valores em faixas/categorias

b) executa estatísticas descritivas por faixa/categoria

```
sIdade
a    40
b    45
c    47
d    37
e    21
f    19
g    30
h    20
i    20
j    40
dtype: int64
```





Método MUITO Útil: cut

Sintaxe: `pandas.cut(x, bins, right=True, labels=None, retbins=False, include_lowest=False)`

Retorna os índices das categorias/faixas (*bins*) de cada valor de x. (pode ser aplicado sobre o *index)*

x – *array* unidimensional a ser dividido em categorias (faixas)

bins = *int ou uma sequência de escalares.*

Se bins é um int, define o número de categorias/faixas nas quais os valores de x serão divididos. Todas as faixas têm a mesma amplitude e o intervalo de x é estendido por 0,1% de cada lado para incluir os valores mínimo ou máximo de x.

Se bins é uma sequência, ela define os limites de cada categoria/faixa. Permite faixas de largura não uniforme. Nenhuma extensão do intervalo de x é feita.

`right` = True – indica se as faixas incluem o limite superior Ex. bins= [1,2,3,4] indicam (1,2), (2,3), (3,4).

`labels` = None ou *array* – se for especificado um *array,* este será usado como *labels* para as categorias/faixas resultantes.

`retbins` = **False**– se True, retorna uma tupla onde o segundo elemento é um *array* com os limites inferiores das faixas.

`include_lowest` = **False** – se True, o primeiro escalar da sequência é incluído no intervalo da primeira faixa.

cut: Primeira forma: bins = *int*

Sintaxe: `pandas.cut(x, bins, right=True, labels=None, retbins=False, include_lowest=False)`

```
sIdade
a    40
b    45
c    47
d    37
e    21
f    19
g    30
h    20
i    20
j    40
dtype: int64
```

```
pd.cut(sIdade,bins=3)
a       (37.667, 47.0]
b       (37.667, 47.0]
c       (37.667, 47.0]
d    (28.333, 37.667]
e    (18.972, 28.333]
f    (18.972, 28.333]
g    (28.333, 37.667]
h    (18.972, 28.333]
i    (18.972, 28.333]
j       (37.667, 47.0]
dtype: category
Categories (3, interval[float64]):
     [(18.972, 28.333] < (28.333,
     37.667] < (37.667, 47.0]]
```

```
pd.cut(sIdade, bins=3,labels=['inf','med','sup'])
a    sup
b    sup
c    sup
d    med
e    inf
f    inf
g    med
h    inf
i    inf
j    sup
dtype: category
Categories (3, object): [inf < med < sup]
```

cut: bins = *sequência de escalares*

Sintaxe: `pandas.cut(x, bins, right=True, labels=None, retbins=False, include_lowest=False)`

```
sIdade
a    40
b    45
c    47
d    37
e    21
f    19
g    30
h    20
i    20
j    40
dtype: int64
```

```
pd.cut(sIdade, bins=[30,40,50,80])

a    (30, 40]
b    (40, 50]
c    (40, 50]
d    (30, 40]
e         NaN
f         NaN
g         NaN
h         NaN
i         NaN
j    (30, 40]
dtype: category
Categories (3, interval[int64]):
          [(30, 40] < (40, 50] < (50, 80]]
```

```
pd.cut(sIdade, bins=[30,40,50,80],
                labels=['inf','med','sup'])

a    inf
b    med
c    med
d    inf
e    NaN
f    NaN
g    NaN
h    NaN
i    NaN
j    inf
dtype: category
Categories (3, object): [inf < med < sup]
```

Exemplos: Categorias de Idade e Categorias de Notas

```
sIdade
a    40
b    45
c    57
d    37
e    21
f    19
g    30
h    20
i    20
j    40
dtype: int64
```

```
sNota
a    10
b     3
c     6
d     6
e    10
f     9
g     7
h     8
i     5
j     7
dtype: int64
```

```
cCatI
a    adulto
b    adulto
c     idoso
d    adulto
e     jovem
f     jovem
g    adulto
h     jovem
i     jovem
j    adulto
dtype: category
Categories (3, object):
[jovem < adulto < idoso]
```

```
cCatN
a     ot
b     bx
c     bx
d     bx
e     ot
f     ot
g    med
h    med
i     bx
j    med
dtype: category
Categories (3, object):
     [bx < med < ot]
```

```
cCatI=pd.cut(sIdade,
        bins=[0,21,50,sIdade.max()],
        labels=['jovem','adulto','idoso'])
```

```
cCatN=pd.cut(sNota,
        bins=[0,6,8,sNota.max()],
        labels=['bx','med','ot'],
        Include_lowest=True)
```

Aplicando uma função sobre valores da Series

Aplicando função sobre elementos de uma Series

Sintaxe: `series.apply(função, args=(...))`

Aplica a função nos valores da Series, retornando uma nova Series. ***Função*** pode ser do Python ou definida pelo programador que opere sobre valores individuais da series.
args = (...) argumentos opcionais fornecidos à função

s:

33A	6.324555
33B	3.872983
33C	4.242641
33E	NaN
33A	6.782330

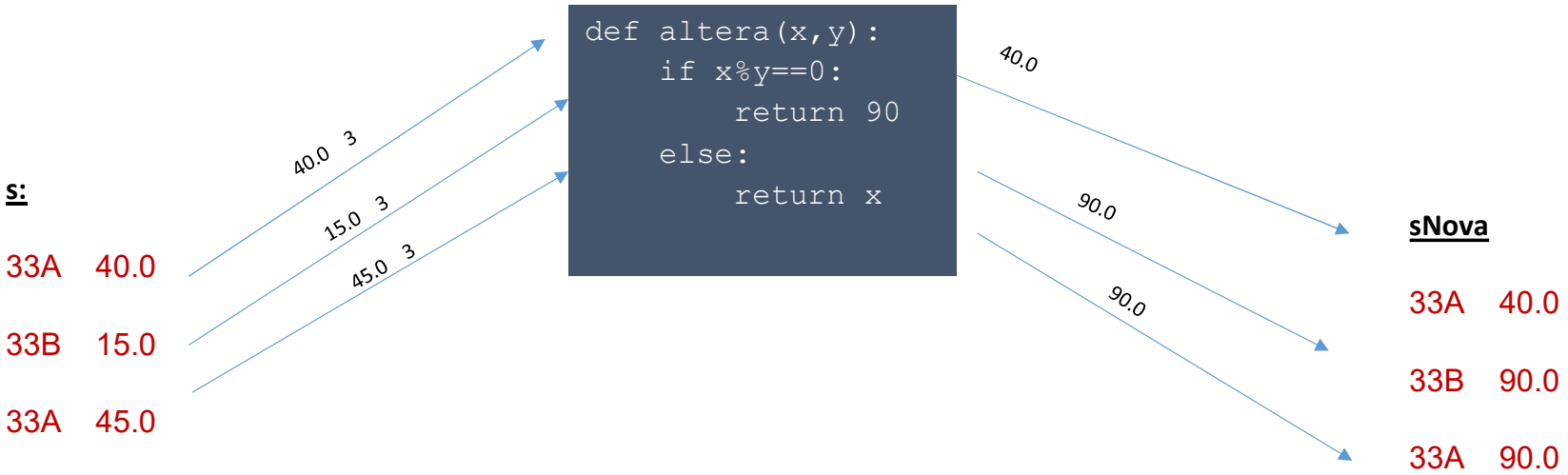
```
import math
s.apply('{:.2f}'.format))
33A     6.32
33B     3.87
33C     4.24
33E      NaN
33A     6.78
dtype: : object
```

s:

33A	40.0
33B	15.0
33C	18.0
33E	NaN
33A	46.0

```
def altera(x,y):
    if x%y==0:
        return 90
    else:
        return x
# torna 90 os múltiplos de 3
s.apply(altera,args=(3,))
33A     40.0
33B     90.0
33C     90.0
33E      NaN
33A     46.0
dtype: float64
```

Simulação do funcionamento : s.apply(altera,args=(3,))



sNova = pd.Series()

Para cada elemento de s

novoValor= resultado da aplicação a função altera sobre o valor do elemento

sNova.loc[índice do elemento] = novoValor

Descrição e Sumarização

Descrição e Sumarização

Medidas de Tendência Central ou Posição

Média:
`series.mean()`[1]
	`>>>s.mean()`
	`31.428571428571427`
	~~`>>>s.mean(level=0)`~~
	~~`a 28.333333`~~
	~~`b 22.500000`~~
	~~`c 45.000000`~~
	~~`dtype: float64`~~

Mediana:
`series.median()`[1]
	`>>>s.median()`
	`30.0`
	~~`>>>s.median(level=0)`~~
	~~`a 30.0`~~
	~~`b 22.5`~~
	~~`c 45.0`~~
	~~`dtype: float64`~~

Moda: `series.mode()`	`print(s.mode())` `0 30`

s:
a 10
b 30
c 50
a 30
b 15
c 40
a 45

~~1 – Com level = 0, operação agrupada por índice~~

NÃO É MAIS VÁLIDO: usar groupby(level=0)

Descrição e Sumarização

Medidas de Tendência Central ou Posição

Máximo: `series.max()` [1][2] Mínimo: `series.min()`[1][2] Índice 1º Mínimo: `series.`idxmin() Índice 1º Máximo: `series.`idxmax()	`>>>s.max() - s.min()` 40 #(50 – 10) ~~`>>>s.max(level=0)`~~ ~~a 45~~ ~~b 30~~ ~~c 50~~

Quantil:
`series.quantile (q=%)`
padrão q=0.5
	`>>>s.quantile()`
	`30.0`
	`print(s.quantile(0.9))`
	`47.0`

s:

a	10
b	30
c	50
a	30
b	15
c	40
a	45

q:

a	1
b	3
c	5
a	3
b	1
c	4
a	4

1 – ~~Com level = 0, operação agrupada por índice~~
2 – Operação aceita no atributo index

Descrição e Sumarização
Medidas de Dispersão

Amplitude: val max - val min	`>>>s.max() - s.min()` 40 #(50 – 10)
Variância: `series.var()` [1]	`>>>s.var()` 222.61904761904762
Desvio Padrão: `series.std()` [1]	`>>>s.std()` 14.920423841803142
Covariância: `series.cov(series)`	`>>>s.cov(q)` 22.5
Correlação: `series.corr(series)`	`>>>s.corr(q)` 0.98721777257162646

s:

a	10
b	30
c	50
a	30
b	15
c	40
a	45

q:

a	1
b	3
c	5
a	3
b	1
c	4
a	4

~~1 – Com level = 0, operação agrupada por índice~~
2 – Operação aceita no atributo index

Descrição e Sumarização

Totalizações

Soma:

`series.sum()` [1]

```
>>>s.sum()
220
>>>s.sum(level=0)
a  85
b  45
c  90
```

Quantidade:

`series.count()`

```
>>>s.count()
7
```

Contagem de valores exclusivos:

`series.value_counts()` [2]

(Tabela de frequências)

Ordenada decrescentemente pela quantidade de ocorrências
Exclui valores de NA por padrão
Alguns Parâmetros – Todos Opcionais

- normalize: boolean, default False. Se True, frequências relativas
- sort: booleano, padrão True. Classificar ou não
- ascending: em ordem crescente de valores. , padrão False.
- dropna: booleano, padrão True. Não inclui contagens de NaN.

```
>>>s.value_counts()
30  2
15  1
45  1
10  1
50  1
40  1
dtype: int64

>>>s.index.value_counts()
a  3
c  2
b  2
dtype: int64
```

s:

a	10
b	30
c	50
a	30
b	15
c	40
a	45

q:

a	1
b	3
c	5
a	3
b	1
c	4
a	4

~~1 – Com level = 0, operação agrupada por índice~~
2 – Operação aceita no atributo index

Descrição e Sumarização

Resumo

Resumo:

```
series.describe()
```

```
>>>s.describe()

count      7.000000
mean      31.428571
std       14.920424
min       10.000000
25%       22.500000
50%       30.000000
75%       42.500000
max       50.000000
dtype: float64
```

s:

a	10
b	30
c	50
a	30
b	15
c	40
a	45

Visualização Gráfica

Métodos para Gráficos no Pandas

Sintaxe:

```
series.plot(kind='line', figsize=None, title=None, legend=False, … )
```

kind= 'line' : linha (default) 'bar' : barra vertical 'barh' : barra horizontal
'hist' : histograma 'box' : boxplot 'area' : area plot 'pie' : pizza
entre outros

figsize = *(altura,largura)* em polegadas

***title* =** título do gráfico

legend= o que representa o eixo x

Há métodos específicos para os gráficos mais utilizados:

- ✓ de linha - serie.plot.line()
- ✓ de barra - serie.plot.bar()
- ✓ histograma - serie.plot.hist()
- ✓ de pizza - serie.plot.pie()
- ✓ de dispersão - serie.plot.scatter() *(apenas para DataFrame)*

Gráficos Usuais

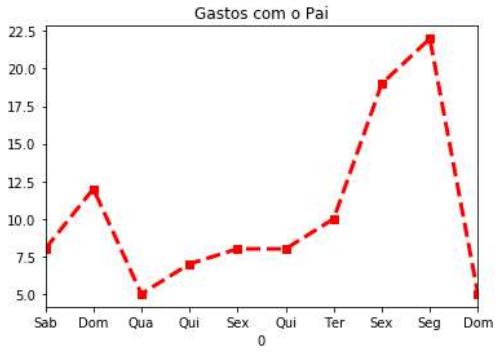
Atributos comuns	• title="título", padrão '' • figsize=(a,l), padrão None • legend=True/False padrão False

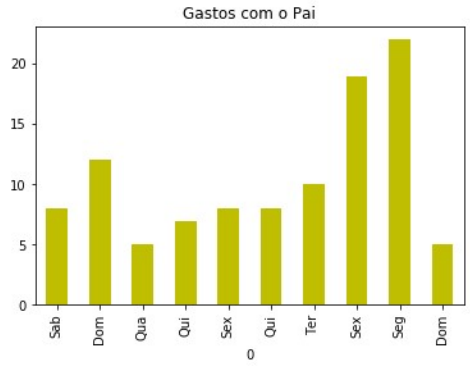
Valores do eixo x: os índices da Series
Valores do eixo y: os valores da Series

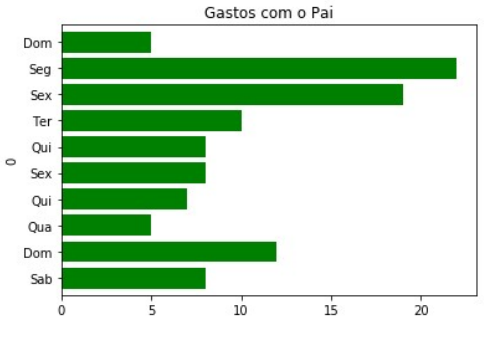
Linha	`series.plot() ou series.plot.line()`	Desenha um gráfico de linha Principais atributos ajustáveis: • color – a cor e pode ser r(red), b(blue), k(black)... • linestyle – o formato da linha. Não contínua: '-'. • linewidth – espessura da linha. • marker – o formato dos pontos: 's' (square) - quadrados, '^'- triângulos, '*',etc.
Barra	`Vertical: series.plot.bar()`	Desenha um gráfico de barra Principais atributos ajustáveis: • x - posição das barras no eixo X • y - altura das barras no eixo Y • width - espessura das barras • color - cor
	`Horizontal: series.plot.barh()`	
Pizza	`series.plot.pie()`	Desenha um gráfico de pizza Principais atributos ajustáveis: • autopct="%.1f“ - Valor percentual da faixa

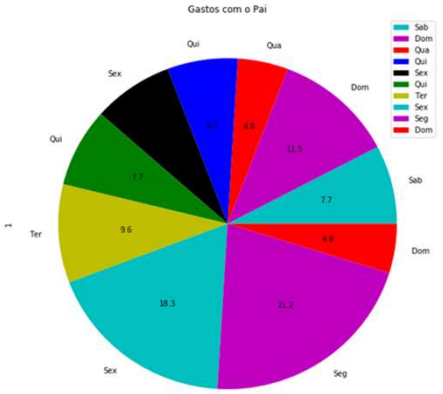
Exemplos

```
sGastos=pd.read_excel("gastosAlimPai.xlsx", header=None,index_col=0, squeeze=True, decimal=',')

sGastos.plot.line(title="Gastos com o Pai", linestyle='--',linewidth=3.0, color='r', marker='s')

sGastos.plot.bar(title="Gastos com o Pai", color='y', width = 0.5)

sGastos.plot.barh(title="Gastos com o Pai", color='g', width = 0.8)

sGastos.plot.pie(title="Gastos com o Pai",colors= ['c','m','r','b','k','g','y'],
                 legend = True, autopct="%.1f",figsize=(10,10))
```









MatPlotLib

A biblioteca ***matplotlib*** do Python é utilizada para a visualização de dados e criação de gráficos 2D. Apresenta uma série de possibilidades gráficas como gráficos de barra, linha, pizza, histogramas, entre muitos outros.

Forma básica para utilizar o MatPlotLib:

- ✓ **1º Passo)** Importar o(s) módulo(s):

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Antes de exibir é possível realizar várias alterações no gráfico como criação da área, traçado dos pontos, mudança do label nos eixos, etc.

- ✓ **2º Passo)** Carregar o conjunto de dados e exibi-los:

```
#Entrega os dados a exibir
plt.plot([0,10,20,30])
#Exibe (Padrão: gráfico de linha)
plt.show()
```