

## Problema 1

O gerente de RH de uma empresa observou que a justificativa fornecida pelos funcionários para suas faltas era "moradia distante do local de trabalho". Para verificar se isto é verdade, solicitou uma análise das distâncias entre as moradias dos faltosos e o local de trabalho. As distâncias devem ficar distribuídas em 10 grupos, um para cada um dos intervalos:

(0 a 1Km], (1km a 3km], (3km a 5km], ..., (19km a 21km], (21km,  $\infty$ )

Esta empresa tem os seguintes dados de seus funcionários em um arquivo:

- ✓ matrícula, nome, endereço, distância ao local de trabalho, data de admissão

e a frequência, em outro arquivo, que contém os seguintes dados:

- ✓ matrícula, data, frequência (1 – presente, 0 – ausente)

1

## Análise da Solução

1) Coletar os dados: Filtrar distância de faltosos (por ocorrência)

Cadastro				Frequência			Distância Faltas		
matr	nome	Endereço	Distância em Km	matr	Dia	Freq	matr	Distância em Km	Falta
210	Zé Carioca	Rua A, 01	0,5	210	2/1	0	210	0,5	2/1
211	Pedrinho	Rua D, 10	0,9	211	2/1	0	211	0,9	2/1
212	Narizinho	Rua E, 34	1,0	212	2/1	1	212	2,5	2/1
213	Emília	Rua E, 34	1,0	213	2/1	1	210	0,5	3/1
214	Visconde	Rua E, 34	1,0	214	2/1	1	212	1,0	3/1
215	Zezinho	Rua W, 23	4,5	215	2/1	1	210	0,5	3/1
216	Huguinho	Rua W, 23	4,5	...	...	...	253	4,3	3/1
230	Luisinho	Rua W, 23	4,5	...	...	...	212	1,0	4/1
231	Gastão	Rua F, 13	2,5	218	3/1	0	232	3,0	4/1
232	Cebolinha	Rua M, 33	8,9	211	3/1	1	210	0,5	5/1
233	Lalá	Rua J, 13	2,1	212	3/1	0	211	1,0	4/1
234	Lelé	Rua J, 13	2,1	213	3/1	1	251	4,2	5/1
235	Lili	Rua J, 13	2,1	...	...	...	233	2,1	6/1
241	Clarabela	Rua O, 23	22,8	299	3/1	0	231	2,1	8/1
251	Magali	Rua T, 21	4,2	210	4/1	1	234	2,9	9/1
252	Donald	Rua J, 21	5,0	211	4/1	0	252	5,0	9/1
253	Margarida	Rua K, 33	4,3	...	...	...	282	6,0	10/1
....				...	...	...	...		

2

## Análise dos Dados

### 2) Agrupar e Totalizar os dados

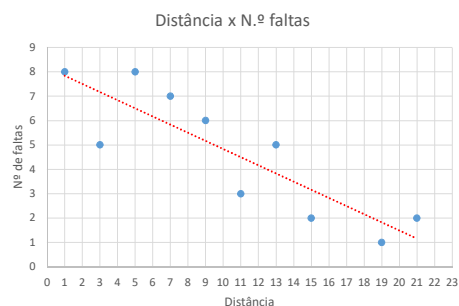
matr	dist	faltas
210	0,5	9,0
211	0,9	2,0
231	2,5	8,0
210	0,5	9,0
241	1,0	9,0
210	0,5	3,0
251	4,3	8,0
210	1,0	9,0
241	3,0	6,0
210	0,5	5,0
211	1,0	6,0
251	4,2	9,0
241	2,1	6,0
231	2,1	8,0
251	2,9	8,0
251	5,0	9,0
241	6,0	10,0
...		



x	y
Distância	N.º faltas
1,0	8
3,0	5
5,0	8
7,0	7
9,0	6
11,0	3
13,0	5
15,0	2
19,0	1
21,0	2



### 3) Visualizar os dados



O gráfico mostra uma associação, de sentido contrário, entre o número de faltas e a distância. Assim, quanto maior é a distância de casa, menor é a tendência para faltar!

3

## Problema 2

Os pais de um bebê recém nascido que dividem seu atendimento noturno acham que o outro está mais descansado. Cada um registrou as horas de sono durante 30 dias e desejam compará-las.

Registro da mãe

8,7	9,3	4,0
9,4	5,3	7,4
6,6	7,3	6,3
6,0	6,7	5,9
6,9	5,8	10,0
5,4	4,7	6,5
5,3	5,6	8,6
8,9	5,9	7,7
10,1	5,4	9,0
9,6	7,6	7,9



Registro do pai

7,1	9,5	7,1
8,3	7,1	7,4
7,1	7,5	7,4
7,9	7,9	7,8
7,5	6,4	6,2
6,2	6,2	8,6
8,2	7,5	8,4
8,7	7,7	6,6
8,5	7,6	8,1
7,6	8,8	7,1

Como compará-las?

4

## Análise dos Dados

Registro da mãe

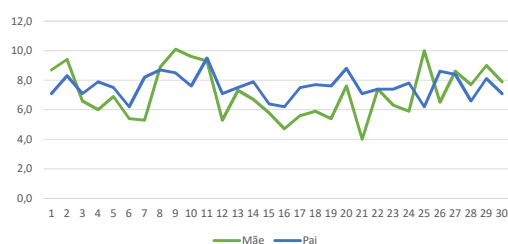
8,7	9,3	4,0
9,4	5,3	7,4
6,6	7,3	6,3
6,0	6,7	5,9
6,9	5,8	10,0
5,4	4,7	6,5
5,3	5,6	8,6
8,9	5,9	7,7
10,1	5,4	9,0
9,6	7,6	7,9

Registro do pai

7,1	9,5	7,1
8,3	7,1	7,4
7,1	7,5	7,4
7,9	7,9	7,8
7,5	6,4	6,2
6,2	6,2	8,6
8,2	7,5	8,4
8,7	7,7	6,6
8,5	7,6	8,1
7,6	8,8	7,1



Horas de Sono



Não esclarecedor...

5

## Análise dos Dados

Registro da mãe

8,7	9,3	4,0
9,4	5,3	7,4
6,6	7,3	6,3
6,0	6,7	5,9
6,9	5,8	10,0
5,4	4,7	6,5
5,3	5,6	8,6
8,9	5,9	7,7
10,1	5,4	9,0
9,6	7,6	7,9



Registro do pai

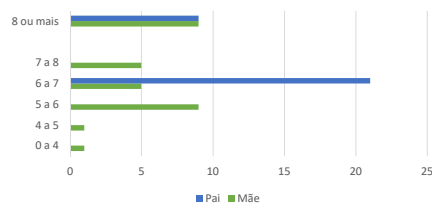
7,1	9,5	7,1
8,3	7,1	7,4
7,1	7,5	7,4
7,9	7,9	7,8
7,5	6,4	6,2
6,2	6,2	8,6
8,2	7,5	8,4
8,7	7,7	6,6
8,5	7,6	8,1
7,6	8,8	7,1

Dias por intervalo

Tempo(em h)	Mãe	Pai
0 a 4	1	0
4 a 5	1	0
5 a 6	9	0
6 a 7	5	21
7 a 8	5	0
8 ou mais	9	9



Comparativo de Horas de Sono (em dias)



Resumo	Mãe	Pai
Media	7,1	7,6
Moda	5,4	7,1
Mediana	6,8	7,6
Variância	2,8	0,7
Desvio Padrão	1,7	0,8

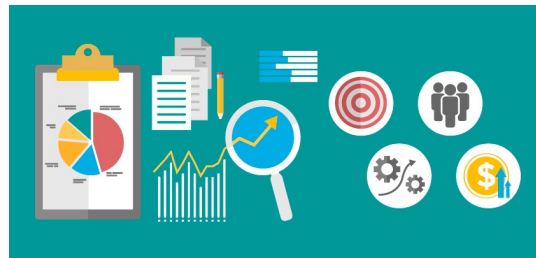
6

## Análise Estatística de Dados

A maioria das áreas de conhecimento, sobretudo áreas de ciências sociais, biológica, saúde e negócios, recorrem à análise de dados para confirmar e/ou propor novas teorias.

Os dados são instrumentos essenciais à compreensão do mundo.

A análise de dados busca descrever e resumir dados, identificar relações e diferenças entre variáveis, comparar variáveis e fazer previsões.



7

## Análise Estatística de Dados: exemplos

- A. a partir de resultados experimentais (dados), descrever uma lei que os explique
- B. a partir de respostas de um questionário sobre um produto (dados), mensurar o grau de satisfação com o produto e melhorias desejadas pelos mesmos;
- C. a partir de características de composição de alimentos, como teores de calorias, glicídios, proteínas, lipídios, cálcio, fósforo, ferro (dados), encontrar graus de similaridade ou adequação para dietas específicas em determinadas doenças;
- D. a partir do monitoramento de reações de grupos distintos em diferentes situações simuladas via realidade virtual(dados), encontrar um padrão de comportamento;
- E. a partir de indicadores socioeconômicos por micro áreas (dados), caracterizar regiões urbanas em relação ao risco à saúde.
- F. a partir do monitoramento do tráfego virtual (dados), determinar o impacto de uma notícia em diferentes perfis

8

## Análise Estatística de Dados

- É uma técnica analítica, com objetivo definido, pela qual se dá ordem, estrutura e significado aos dados, que podem ser quantitativos ou qualitativos.
- Utiliza-se de tecnologias da informação para tratar e transformar grandes volumes de dados, buscando identificar respostas, soluções, tendências, regras, padrões, diferenças e variações a partir de um volume de dados.
- Na primeira fase, conhecida como análise exploratória, onde os dados são organizados em tabelas e gráficos e deseja-se resumos descritivos como a média, a mediana, a moda, o desvio padrão, entre outras, a tecnologia permite o armazenamento organizado dos dados, cálculo automático dos resumos e várias formas de visualização.
- Na fase seguinte, onde se inferem possíveis modelos de comportamento, distribuição e agrupamento, entre outros, a tecnologia fornece várias ferramentas que os interpretam, incluindo operações como a identificação de divergências ou de padrões.

9

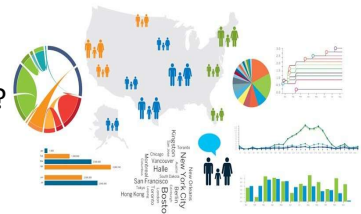
## Análise Exploratória de Dados

Objetivo: examinar os dados antes da aplicação de qualquer técnica estatística ou de mineração de dados.

Ajuda a extrair informações relevantes de um conjunto de dados e a entender os dados e as relações que existem entre as variáveis analisadas.

Utilizando tabelas e gráficos, permite responder questões como:

- Os dados são quase todos iguais ou muito diferentes uns dos outros?
- Existe algum padrão subjacente ou alguma tendência?
- Existem alguns agrupamentos especiais?
- Existem alguns dados muito diferentes da maior parte?



Estas questões, de um modo geral, são muito difíceis de serem respondidas observando os dados brutos, não organizados.

10

## Classificação dos Dados

Os dados observados (variáveis) podem ser classificados quanto a sua natureza como:

Variáveis		Tipos	Descrição	Exemplos	Gráficos Usuais
Qualitativas ou Categóricas	separados em diferentes categorias que se distinguem por alguma característica não-numérica	Nominal	Não existe nenhuma ordenação	Cor dos olhos, sexo, estado civil, tipo sanguíneo, fumante/não fumante	Pizza Colunas Barras Linhas
		Ordinal	Existe uma ordenação I,II,III	•Nível de escolaridade, estágio da doença, colocação de concurso, mês de observação (jan,fev,...), classe social (alta, média, baixa), desempenho (ruim, regular, bom).	
Quantitativas	números que representam contagens ou medidas com níveis de mensuração intervalar ou de razão	Discretas	Valor pertence a um conjunto enumerável	Número de filhos por casal, quantidade de leitos, número de cômodos em um domicílio, número de bactérias por litro de leite, número de cigarros fumados por dia, número de reprovações, "idade em anos	Barras Colunas Diagrama de Dispersão Linha
		Contínuas	Valor pertence a um intervalo real	Medidas de altura e peso, taxa de glicose, nível de colesterol, tempo, renda per capita, "nota na prova", "pontuação no vestibular", "altura do entrevistado".	

Adaptado de Análise Exploratória de Dados\*

\*[http://www.uel.br/pos/estatisticaquantitativa/textos\\_didaticos/especializacao\\_estadistica.pdf](http://www.uel.br/pos/estatisticaquantitativa/textos_didaticos/especializacao_estadistica.pdf)

11

## Etapas na Análise de Dados Quantitativos

- I. Organizar os Dados
- II. Agrupar e resumir os dados através de tabelas de frequências, entre outros, realizando exames gráficos que permitam entendê-los
- III. Resumir as principais estatísticas como, por exemplo, medidas de tendência central e/ou de dispersão
- IV. Analisar e interpretar os dados por meio de cruzamentos de tabelas, análise de correlação, entre outros

12

## Distribuição de Frequência

Agrupar os dados em classes e contabilizar o número de ocorrências em cada classe.

Importante quando há uma grande quantidade de dados pois apresenta os dados de modo mais conciso, facilitando a visualização e cálculos.

Tanto os dados qualitativos quanto os quantitativos podem e devem ser agrupados em frequências.

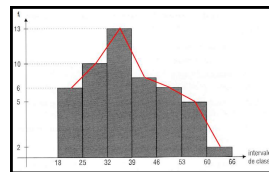
Em dados quantitativos contínuos, divide-se a faixa de variação dos dados em intervalos de classes e a distribuição de frequência é realizada nestes intervalos.

**Histograma:** Representação gráfica da distribuição de frequências de um conjunto de dados (um gráfico de barras verticais ou horizontais).

**Polígonos de Frequência:** Semelhante ao histograma, mas construído a partir dos pontos médios das classes

18
19
25
31
18
24
18
23
26
22
...
63

Classes	Intervalos das classes	f
1	18 — 25	6
2	25 — 32	10
3	32 — 39	13
4	39 — 46	8
5	46 — 53	6
6	53 — 60	5
7	60 — 66	2
<b>Somas</b>		<b>50</b>



<https://www.gestaodessegurancaprivada.com.br/histograma-de-frequecia-conceito/>

<http://professorguru.com.br/estatistica/gr%C3%A1ficos%20estat%C3%ADsticos/exemplo%20de%20histograma%20e%20pol%C3%ADgono%20de%20frequ%C3%Aancias.html> 13

## Variáveis quantitativas

**Medidas de Tendência Central ou de Posição:** valor ao redor do qual os dados estão distribuídos.

- Máximo: a maior observação
- Mínimo: a menor observação
- Moda: é o valor (ou categoria) que ocorre com maior frequência.
- Média: soma de todos os valores da variável/número de observações.
- Mediana: valor que deixa 50% das observações à sua esquerda
- Quartis: divide um conjunto de valores dispostos em forma crescente em quatro partes.
  - ✓ Primeiro Quartil (Q1): valor que deixa 25% das observações à sua esquerda.
  - ✓ Terceiro Quartil (Q3): valor que deixa 75% das observações à sua esquerda.

## Variáveis quantitativas

**Medidas de Dispersão:** valores que resumem a variabilidade de um conjunto de dados

- Amplitude: diferença entre o valor máximo e o valor mínimo
- Variância: média dos quadrados dos desvios em relação à média aritmética
- Desvio Padrão: mede a variabilidade independente do número de observações na mesma unidade de medida da média
- Coeficiente de Variação: mede a variabilidade percentual independente da unidade de medida ou da ordem de grandeza da variável

15

# Excel



# Pandas

## 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  . . . Apelido do módulo

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

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

19

## 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:

0	1	2	3	<span style="border: 1px solid blue; padding: 2px;">Índices</span>
6.0	10.0	9.5	6.0	<span style="border: 1px solid blue; padding: 2px;">Elementos</span>

	<span style="border: 1px solid blue; padding: 2px;">Índices</span>	P1	P2	P3	PF
1		6.0	10.0	9.5	6.0
2		7.0	6.0	9.0	8.0

20

## 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

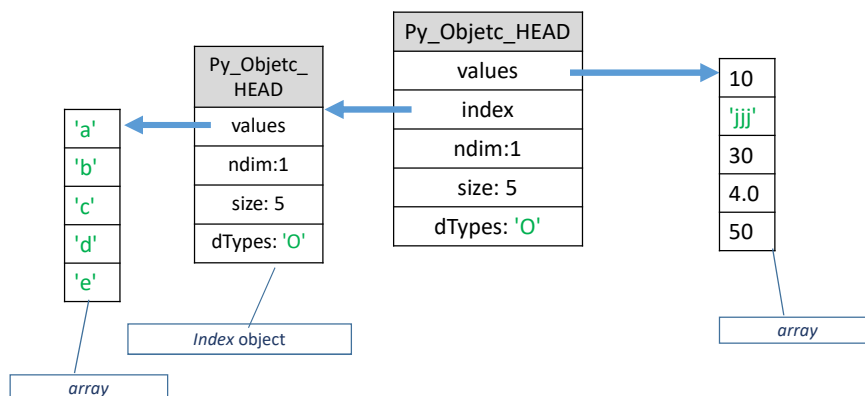
Índice Dado

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

21

## Esquema simplificado do objeto Series



22



## Construindo uma *Series*

sem especificação dos índices

Sintaxe: `pd.Series (valores)`

# a partir de uma lista

```
>>>s1=pd.Series([4, 7, -5])
```

```
>>>s1
```

```
0    4
```

```
1    7
```

```
2   -5
```

```
dtype: int64
```

Índice criado automaticamente

# a partir dos caracteres de uma string

```
>>>s2=pd.Series(list('ab'))
```

```
>>>s2
```

```
0    a
```

```
1    b
```

```
dtype: object
```

# a partir de um intervalo

```
>>>s3=pd.Series(range(4,10,3))
```

```
>>>s3
```

```
0    4
```

```
1    7
```

```
dtype: int32
```

# a partir de um escalar

```
>>>s4= pd.Series(5)
```

```
>>>s4
```

```
0    5
```

```
dtype: object
```

# a partir de um dicionário

```
>>>s5= pd.Series({ 69:'E' , 65:'A' })
```

```
>>>s5
```

```
65    A
```

```
69    E
```

```
dtype: object
```

Ordenado por chave

23



## Mãos na Massa

1. A área (Kb) necessária para armazenar uma foto em alta definição depende do formato do arquivo. Considerando os seguintes valores para uma foto 10x15 no *Paint*: jpg:665, png:3348, bmp:9217, gif:1118, tif:3599, construir:
  - a) a *Series* *sF1\_l*, com os valores destas áreas;
  - b) um dicionário com estes dados. Criar a *Series* *sF1\_d* a partir do dicionário.
2. Uma disciplina oferta 5 turmas ('33A', '33C', '33F', '33B', '33E'), de 50 vagas, em horários distintos (representados pela letra da turma: A é mais cedo que B que é mais cedo que C, ...) e 2 turmas para os alunos de currículos antigos ('A33', 'A44') com 40 vagas. Após o primeiro dia de matrícula, a ocupação das turmas foi armazenada no seguinte dicionário:
 

```
{ '33A':40, '33C':40, '33F':16, '33B':15, '33E':None, 'A44':None, 'A33':25 }
```

 A partir deste dicionário, crie uma *Series*.

24



## Mãos na Massa

```
import pandas as pd
sF1_l = pd.Series([9217, 1118, 665, 3348, 3599])
print("Series 1:\n",sF1_l)

df1Paint={'jpg':665,'png':3348,'bmp':9217,'gif':1118,'tif':3599}
sF1_d = pd.Series(df1Paint)
print ("Series 2:\n", sF1_d)
```

"A área (Kb) necessária para armazenar uma foto em alta definição depende do formato do arquivo. Considerando os seguintes valores para uma foto 10x15 no Paint: jpg:665, png:3348, bmp:9217, gif:1118, tif:3599, construir:

- a) a *series sF1\_l*, com estes valores;
- b) um dicionário com estes dados e carregá-lo para a *series, sF1\_d*."

```
Series 1:
0    9217
1    1118
2     665
3    3348
4    3599
dtype: int64
```

```
Series 2:
bmp    9217
gif    1118
jpg     665
png    3348
tif    3599
dtype: int64
```

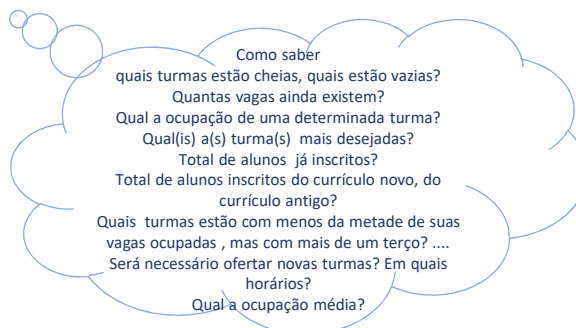
25



## Mãos na Massa

```
import pandas as pd
dInsc={'33A':40,'33C':40,'33F':16,'33B':15,'33E':None,'A44':None,'A33':25}
sInsc=pd.Series(dInsc)
print('Inscrições 1º dia:\n', sInsc)
```

```
Inscrições 1º dia:
33A    40.0
33B    15.0
33C    40.0
33E     NaN
33F    16.0
A33    25.0
A44     NaN
dtype: float64
```



"Uma disciplina oferta 5 turmas ('33A', '33C', '33F', '33B', '33E'), de 50 vagas, em horários distintos (representados pela letra da turma: A é mais cedo que B que é mais cedo que C, ...) e 2 turmas para os alunos de currículos antigos ('A33', 'A44') com 40 vagas. Após o primeiro dia de matrícula, a ocupação das turmas foi armazenada no seguinte dicionário: {'33A':40,'33C':40,'33F':16,'33B':15,'33E':None,'A44':None,'A33':25}. A partir deste dicionário, crie uma Series."

26



## Valores, índices e tamanho

Valores: `series.values`

Índices: `series.index`

Tamanho: `series.size`

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

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

```
>>>sF1_l.values
array([9217, 1118,  665, 3348, 3599],
      dtype=int64)

>>>print(sF1_l.values)
[9217 1118  665 3348 3599]

>>>sF1_l.index
RangeIndex(start=0,stop=5,step=1)

>>>sF1_l.size
5
```

```
>>>sF1_d.values
array([9217, 1118,  665, 3348, 3599],
      dtype=int64)

>>>print(sF1_d.values)
[9217 1118  665 3348 3599]

>>>sF1_d.index
Index(['bmp', 'gif', 'jpg', 'png',
      'tif'], dtype='object')

>>>sF1_d.size
5
```

27



## Construindo uma Series

com especificação dos índices

Sintaxe:

```
pd.Series(valores , index = array unidimensional )
```

# a partir de uma lista

```
>>>l= [ 4,7,-5 ]
>>>s7= pd.Series( l , index=['a','c','b'])
>>>s7
a    4
c    7
b   -5
dtype: int64
```

# a partir de um dicionário

```
>>>vogais=['vog1', 'vog5', 'vog2']
>>>s8= pd.Series({'vog1':'A', 'vog3':'I', 'vog5':'U'}, index = vogais)
>>>s8
vog1    A
vog5    U
vog2    NaN
dtype: object
```

pois índice  
≠ lista de chaves

```
'vog1'
'vog5'
'vog2'
```

28

## Mãos na Massa

Crie as seguintes Series que armazenam os gastos totalizados por dia da semana com alimentação:

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

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

```
SGnum = pd.Series([10.0,23.0,22.4,10.0,15.0,12.0,25.0])
```

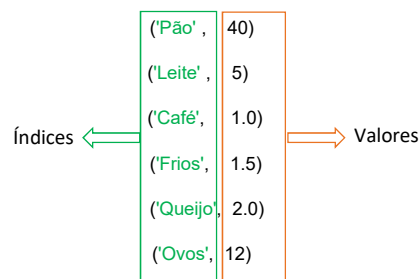
```
sGdia=pd.Series([10.0,23.0,22.4,10.0,15.0,12.0,25.0],
                index=['Seg', 'Ter', 'Qua', 'Qui', 'Sex', 'Sab', 'Dom'])
```

29

## Mãos na Massa

Crie a Series sCompras, com os produtos (e respectivas quantidades) usados no café da manhã que estão na seguinte lista de tuplas:

```
[('Pão',40),('Leite',5),('Café',1.0),('Frios',1.5),('Queijo',2.0),('Ovos',12)]
```



30

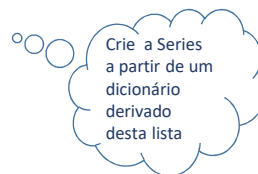


## Uma Solução

"Crie a Series sCompras, com os produtos (e respectivas quantidades) usados no café da manhã que estão na seguinte lista de tuplas: [('Pão',40),('Leite',5),('Café',1.0),('Frios',1.5),('Queijo',2.0),('Ovos',12)]"

```
import pandas as pd
l=[('Pão',40),('Leite',5),('Café',1.0),('Frios',1.5),('Queijo',2.0),('Ovos',12)]
lprod=[]
lqt=[]
for (prod,qt) in l:
    lprod.append(prod)
    lqt.append(qt)
sCompras=pd.Series(lqt, index=lprod)
print(sCompras)
```

Pão	40.0
Leite	5.0
Café	1.0
Frios	1.5
Queijo	2.0
Ovos	12.0
dtype: float64	



31



## Uma Solução

"Crie a Series sCompras, com os produtos (e respectivas quantidades) usados no café da manhã que estão na seguinte lista de tuplas: [('Pão',40),('Leite',5),('Café',1.0),('Frios',1.5),('Queijo',2.0),('Ovos',12)]"

```
import pandas as pd
l=[('Pão',40),('Leite',5),('Café',1.0),('Frios',1.5),('Queijo',2.0),('Ovos',12)]
dCompras=dict(l)
sCompras=pd.Series(dCompras)
print(sCompras)
```

Café	1.0
Frios	1.5
Leite	5.0
Ovos	12.0
Pão	40.0
Queijo	2.0
dtype: float64	

32





## Mãos na Massa

Crie a Series SF2, com as áreas mínimas necessárias para salvar uma foto 10x15 via Paint em cada tipo de formato. Utilize os índices e os valores de sF1\_d, para construir sF2.

```
>>>sF2=pd.Series(sF1_d.values,index=sF1_d.index)
>>>sF2
bmp      9217
gif      1118
jpg       665
png     3348
tif     3599
dtype: int64
```

É o mesmo array de dados de sF1\_d

```
>>>sF2c=pd.Series(sF1_d.values,index=sF1_d.index,copy=True)
>>>sF2c
bmp      9217
gif      1118
jpg       665
png     3348
tif     3599
dtype: int64
```

É outro array de dados com mesmos valores de sF1\_d

Alternativa: usar o método `series.copy()`:  
`sF2c=sF1_d.copy()`

33



## Construindo uma Series

a partir de um arquivo Excel

**Sintaxe:** `pd.read_excel(caminho, index_col= n, squeeze=True, header=None, decimal=',')`

**caminho** - localização do arquivo: composto pelo caminho (absoluto/relativo) e nome

**index\_col = n** - em geral 0. O número da coluna do arquivo 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

**header = None** - para arquivos que não possuem linha de cabeçalho

**decimal = ','** - quando o separador de casas decimais é a vírgula, Padrão: '.'

C:/dt/a1.xlsx

	A	B	C
1	usu10	43,00	
2	usu18	61,00	
3	usu14	66,00	

```
sLab=pd.read_excel("C:/dt/a1.xlsx",squeeze=True,header=None,index_col=0,
                    decimal=',')
```

```
print(sLab.head(3))
```

Mostra os *n* 1<sup>os</sup> elementos

```
usu10  43.0
usu18  61.0
usu14  66.0
Name: 1, dtype: float64
```

a2.xls

	A	B
1	0,45	
2	3,67	
3	1,74	

```
sNum=pd.read_excel("a2.xls",squeeze=True,header=None, decimal=',')
print(sNum.head(3))
```

```
0    6.15
1    4.17
2    5.86
Name: 0, dtype: float64
```

Arquivo na mesma pasta do programa .py

34



## Selecionando itens da Series

### Sintaxe:

```
series.loc[indice] ou series.loc[lista de índices]
series.iloc[posição] ou series.iloc[lista de posições]
```

Retorna o valor do elemento indexado por *índice* ou uma nova Series com os elementos da *lista de índices*.  
**.loc** para os índices criados, **.iloc** para a posição no índice

#### 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]'
```

35



## Mãos na Massa

1. Mostre os gastos com alimentação nos dias úteis e no final de semana, separadamente.
2. Uma disciplina oferta 5 turmas ('33A', '33C', '33F', '33B', '33E'), cada uma com 50 vagas e 2 turmas para os alunos de currículos antigos ('A33', 'A44') com 40 vagas. Os horários são organizados por letra: A é mais cedo que B, que é mais cedo que C, ...

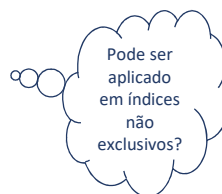
Com a *Series* da ocupação das turmas após o primeiro dia de matrícula criada anteriormente (sInsc), mostre a ocupação das turmas do currículo antigo, das turmas do 1º e 2º horários e da turma do último horário.

36

## Uma Solução

'''Mostre os gastos com alimentação nos dias úteis e no final de semanal separadamente.'''

```
import pandas as pd
sGdia=pd.Series([10.0,23.0,22.4,10.0,15.0,12.0,25.0],
                index=['Seg','Ter','Qua','Qui','Sex','Sab','Dom'])
print('Gasto Alimentação\n')
print('Dias Úteis\n')
print(sGdia.loc['Seg':'Sex'])
print(sGdia.loc[['Sab','Dom']])
```



Gasto Alimentação  
Dias Úteis  
Seg 10.0  
Ter 23.0  
Qua 22.4  
Qui 10.0  
Sex 15.0  
dtype: float64

Final de Semana  
Sab 12.0  
Dom 25.0  
dtype: float64

37

## Uma Solução

'''Uma disciplina oferta 5 turmas ('33A', '33C', '33F', '33B', '33E'), cada uma com 50 vagas e 2 turmas para os alunos de currículos antigos ('A33', 'A44') com 40 vagas. Os horários são organizados por letra: A é mais cedo que B, que é mais cedo que C, ...  
Com a Series da ocupação das turmas após o primeiro dia de matrícula, criada anteriormente, mostre a ocupação das turmas do currículo antigo, das turmas do 1º e 2º horários e da turma do último horário'''

```
import pandas as pd
dInsc={'33A':40,'33C':40,'33F':16,'33B':15,'33E':None,'A44':None,'A33':25 }
sInsc=pd.Series(dInsc)
print('Ocupação\n')
print('das Turmas Antigas\n')
print(sInsc.loc[['A33','A44']])
print('\ndas Turmas do 1º e 2º horários')
print(sInsc.iloc[0:2])
print('\nda Turma do último horário')
print(sInsc.loc['33F'])
```

Ocupação  
das Turmas Antigas  
A33 25.0  
A44 NaN  
dtype: float64

das Turmas do 1º e 2º  
horários  
33A 40.0  
33B 15.0  
dtype: float64

da Turma do último horário  
16.0



38



## Exibindo índice

```
import pandas as pd
dInsc={'33A':40,'33C':40,'33F':16,'33B': 15,'33E':None,'A44':None,'A33':25}
sInsc=pd.Series(dInsc)
print('Ocupação\n')
print('das Turmas Antigas\n')
print( sInsc.loc[['A33','A44']])
print('\ndas Turmas do 1º e 2º horários')
print(sInsc.iloc[0:2])
print('\nda Turma do último horário')
local = sInsc.index.get_loc("33F")
print(local, ' - ', sInsc.index[local], ' - ', sInsc.iloc[local])
```

Como formatar a Saída?

Ocupação  
das Turmas Antigas  
A33 25.0  
A44 NaN  
dtype: float64

das Turmas do 1º e 2º horários  
33A 40.0  
33B 15.0  
dtype: float64

da Turma do último horário  
4 - 33F - 16.0

Retorna:  
a posição do *label* no índice, quando são exclusivos  
um array de booleanos: quando há *labels* repetidos

39



## Iterando sobre a Series

Sintaxe:

```
for (i,val) in series.iteritems():
    corpo
```

Itera sobre os pares (índice, valor).

```
ind=['Pão','Leite','Café','Frios','Queijo','Ovos']
s=pd.Series( [40, 5, 1.0, 1.5, 2.0, 12], index = ind)

#Toda a Series
print('Toda a Series')

for (i,v) in s.iteritems():
    print(' {} - {}'.format(i,v))

#Series criada pelo slice
print('Series criada pelo slice')
for (i,v) in s.iloc[0:2].iteritems():
    print(' {} - {}'.format(i,v))

#Series criada por lista de índices
print('Series criada por lista de índices')
for (i,v) in s.loc[['Pão', 'Café']].iteritems():
    print(' {} - {}'.format(i,v))
```

s:

Pão	40.0
Leite	5.0
Café	1.0
Frios	1.5
Queijo	2.0
Ovos	12.0

Toda a Series

Pão - 40.0  
Leite - 5.0  
Café - 1.0  
Frios - 1.5  
Queijo - 2.0  
Ovos - 12.0

Series criada pelo slice

Pão - 40.0  
Leite - 5.0

Series criada por lista de índices

Pão - 40.0  
Café - 1.0

40



## Formatando a saída do exercício

```
import pandas as pd
def exibe(s):
    for i,v in s.iteritems():
        print('    {} - {}'.format(i,v))
    return
dInsc={'33A':40, '33C':40, '33F':16, '33B':15, '33E':None, 'A44':None, 'A33':25}
sInsc=pd.Series(dInsc)
print('---OCUPAÇÃO---')
print(' Turmas Antigas')
exibe(sInsc.loc[['A33','A44']])
print('\n Turmas do 1º e 2º horários')
exibe(sInsc.iloc[0:2])
print('\n Turma do último horário')
local = sInsc.index.get_loc("33F")
print('{} - {} - {}'.format(local, sInsc.index[local], sInsc.iloc[local]))
```

Como incluir a turma  
33D ou alterar o valor  
da turma A33 ou  
retirar a turma A44?

---OCUPAÇÃO---

Turmas Antigas

A33 - 25.0

A44 - NaN

Turmas do 1º e 2º horários

33A - 40.0

33B - 15.0

Turma do último horário

4 - 33F - 16.0

41



## Alterando/Incluindo valores na Series

### Sintaxe:

```
series.loc[índice]= valor ou
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.

```
>>>dInsc2 = {'33A':4, '33E':None, '33C':1}
>>>sInsc2=pd.Series(dInsc2)
>>>sInsc2
33A    4.0
33C    1.0
33E    NaN
dtype: float64

>>> 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
```

42



## Descartando elementos da Series

### Sintaxe:

```
series.drop (índice ou lista de índices) *
series.dropna () *
```

**drop:** Retorna uma cópia da *series* sem os elementos da lista de índices.

**dropna:** Retorna uma cópia da *series* sem os elementos cujo valor é NaN

\* com **inplace=True**, realiza a operação na Series, não cria uma cópia

```
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
33A 46.0
dtype: float64
```

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

```
>>>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
```

43



## Ordenar Series Pelos Valores: sort\_values

### Sintaxe:

```
series.sort_values (ascending=True) *
```

Retorna uma cópia da *series* ordenada pelos valores

\* com **inplace=True**, realiza a operação na Series, não cria uma cópia

```
s:
33A 40.0
33D 5.0
33C NaN
33E 18.0
33A 48.0
```

```
>>>sOrdC=s.sort_values()
>>>sOrdC
33D 5.0
33E 18.0
33A 40.0
33A 48.0
33C NaN
dtype: float64
```

```
>>>sOrdD=s.sort_values(ascending=False)
>>>sOrdD
33A 48.0
33A 40.0
33E 18.0
33D 5.0
33C NaN
dtype: float64
```

44



## Ordenar Series Pelos índices: sort\_index

**Sintaxe:** `series.sort_index(ascending=True) *`

Retorna uma cópia da *series* ordenada pelos labels do índice

\* com **inplace=True**, realiza a operação na Series, **não** cria uma cópia

```
>>>sOrdC=s.sort_index()
>>>sOrdC
s:
33A 40.0
33D 5.0
33C 18.0
33E NaN
33A 48.0
dtype: float64
```

```
>>>sOrdD=s.sort_index(ascending=False)
>>>sOrdD
33E NaN
33D 5.0
33C 18.0
33A 40.0
33A 48.0
dtype: float64
```

45



## Mãos na Massa

A mãe do Gasparzinho, anotou no arquivo Excel **gastosAlimMae.xlsx** o quanto ele gastou com alimentação durante uma semana. Em cada linha, registrou na 1ª coluna um dia da semana (Seg, Ter,..., Dom) e na 2ª coluna um dos valores gastos com alimentação no dia:

Sab	76,80
Dom	71,22
Qua	65,18
Qui	65,23
Sex	44,58
Qui	41,52
Ter	12,74
Sex	3,64
Seg	35,28
Dom	69,32
Seg	92,32
...	...

a) A partir deste arquivo, crie uma Series usando a 1ª coluna como índice. Exiba os 8 primeiros itens da *Series* criada. Observe que haverá repetição de valores na 1ª coluna e que os dados não estão organizados por dia.

46



## Uma Solução

" a) A partir deste arquivo, crie uma Series usando a 1ª coluna como índice. Exiba os 8 primeiros itens da *Series* criada. Observe que haverá repetição de valores na 1ª coluna e que os dados não estão organizados por dia.

```
import pandas as pd
```

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

```
print(sValGastos.head(8))
```

```
0
Sab    76.80
Dom    71.22
Qua    65.18
Qui    65.23
Sex    44.58
Qui    41.52
Ter    12.74
Sex     3.64
Name: 1,
dtype: float64
```

47



## Mãos na Massa

b) Construa um script que a partir do mesmo arquivo Excel **gastosAlimMae.xlsx**, exiba os gastos do Gasparzinho agrupados por dia.

Sab	76,80
Dom	71,22
Qua	65,18
Qui	65,23
Sex	44,58
Qui	41,52
Ter	12,74
Sex	3,64
Seg	35,28
Dom	69,32
Seg	92,32
...	...

Para isso, construa um dicionário cujas chaves são os dias da semana e os valores, listas de gastos no dia.

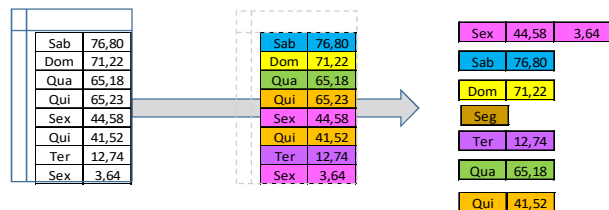
48



## Analizando o problema

### Idéia da Solução:

- I. Criar Series com gastos a partir do arquivo, usando a 1ª coluna como índice
- II. Exibir dados por dia da semana:
  - I. Agrupar gastos por dia da semana:
    - a) Criar um dicionário {dia da semana : []}
    - b) Para cada elemento da Series, incluir o valor na chave do dicionário equivalente ao índice
  - II. Exibir gastos agrupados
    - a) Para cada item do dicionário, exibi-lo.



49

## Uma Solução: Exibir dados por dia da Semana

```
def montaDic(sGastos):
    # Criar dicionário com chave por dia de semana
    dGastos={'Seg':[], 'Ter':[], 'Qua':[], 'Qui':[], 'Sex':[],
            'Sab':[], 'Dom':[]}

    # Incluir gastos no dicionário no respectivo dia
    for (i,v) in sGastos.iteritems():
        dGastos[i].append(v)
    return dGastos

def exibe(sGastos):
    # Agrupar gastos por dia da semana usando um dicionário
    dGastos=montaDic(sGastos)
    # Exibir itens do dicionário por linha
    for (dia,lval) in dGastos.items():
        print('{:s}: '.format(dia),end=' ')
        for v in lval:
            print('{:5.2f}'.format(v),end='\t')
        print(']')
    return
```

50



## Uma Solução: Exibir dados por dia da Semana

```
import pandas as pd
# Criar a Series
sGastos=pd.read_excel("gastosAlimMae.xlsx", header=None,
                     index_col=0, squeeze=True, decimal=',')
# Exibir dados por dia de semana
exibe(sGastos)
```

```
Seg: [ 35.28  92.32  93.87  88.44  85.78  40.28 ]
Ter: [ 12.74  83.15  40.89  55.27  12.74 ]
Qua: [ 65.18  37.35  55.19  48.32  13.84  32.33  42.40 ]
Qui: [ 65.23  41.52  34.91  93.86  85.13  34.91 ]
Sex: [ 44.58   3.64  85.40   5.34  87.96  61.63   5.34 ]
Sab: [ 76.80  56.16  64.50  64.29  45.79  93.80   6.33 ]
Dom: [ 71.22  69.32  93.51  17.89  97.03 ]
```

51



## Salvando uma Series

para um arquivo Excel

### Sintaxe:

```
series.to_excel(caminho, sheet_name = nome, index=True/False, header=False)
```

**caminho** - localização do arquivo: composto pelo caminho (absoluto/relativo) e nome

**sheet\_name = nome** - cria a planilha com o nome fornecido. Padrão = Sheet1. Se o arquivo excel já existe e não há uma planilha com o nome fornecido, a planilha é acrescentada no mesmo arquivo.

**index = True/False** - se True (padrão) salva os índices, um por linha. Se False, não salva.

**header = False** - para não escrever a linha de cabeçalho. Padrão: True arquivos que não possuem linha de cabeçalho

sLab:

```
usu10  43.0
usu18  61.0
usu14  66.0
```

Name: 1, dtype: float64

```
sLab.to_excel("C:/dt/a1.xlsx", header=False)
```

C:/dt/a1.xlsx

	A	B	C
1	usu10	43,00	
2	usu18	61,00	
3	usu14	66,00	

sNum:

```
0  6.15
1  4.17
2  5.86
```

Name: 0, dtype: float64

```
sNum=pd.read_excel("a2.xlsx", header=False, index=False)
```

a2.xlsx

	A	B
1	0,46	
2	3,67	
3	1,74	

Arquivo na mesma pasta do programa .py

52

## Uma Solução: Salvando os valores corrigidos

```
import pandas as pd
def corrige(x):
    if 80<=x<100:
        return x/10
    elif int(x)%10 == 4:
        return x-3
    else:
        return x

# Montar a Series
sGastos=pd.read_excel("gastosAlimMae.xlsx", header=None, index_col=0, squeeze=True, decimal=',')
# Exibir dados por dia de semana
exibe(sGastos)
# Corrigir dados
sGastos=sGastos.apply(corrige)
# Salvar Series
sGastos.to_excel("gastosAlimMaeCor.xlsx", sheet_name="mae", header=False)
```

53

## Visualização

54



## 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.

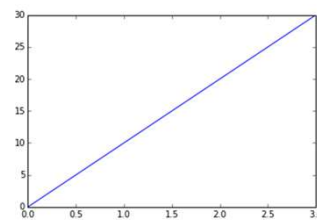
### Forma básica para utilizar o Matplotlib:

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

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

- ✓ **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()
```



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

55



## Traçando gráficos no Pandas

### Sintaxe:

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

**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

**Método nativo do Pandas:** não precisa importar o matplotlib  
Tipos distintos podem requerer diferentes parâmetros.

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

```
sGastos=pd.read_excel("gastosAlimPai.xlsx",
                      header=None,index_col=0, squeeze=True,
                      decimal=',')
sGastos.plot(title="Gastos com o Pai")
```



56



## Métodos para Gráficos no Pandas

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*)

57



## plot.line: gráfico de linha

Principais atributos ajustáveis:

- `color` – a cor e pode ser `r`(red), `b`(blue), `k`(black)...
- `linestyle` – o formato da linha. Para que a mesma não seja contínua define-se como `'--'`.
- `linewidth` – espessura da linha.
- `marker` – o formato dos pontos: `'s'` (square) - quadrados, `'^'` - triângulos, `'*'`, etc.

```
serie = pd.Series([15, 40, 75, 90], index=[10, 20, 30, 40])
serie.plot(linestyle='--', color='r', marker='s', linewidth=3.0)
```

```
sGastos=pd.read_excel("gastosAlimPai.xlsx",
                      header=None, index_col=0, squeeze=True,
                      decimal=',')
sGastos.plot.line(title="Gastos com o Pai", linestyle='--',
                  color='r', marker='s',
                  linewidth=3.0)
```



58

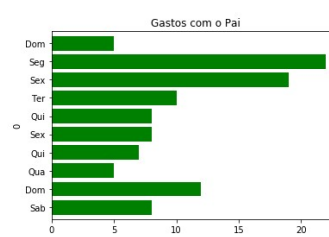
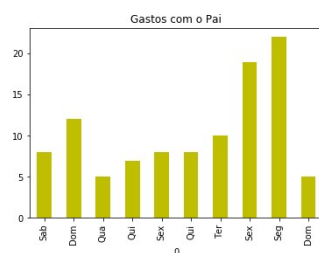


## plot.bar: barras verticais plot.barh: barras horizontais

Principais atributos ajustáveis:

- x - posição das barras no eixo X
- y - altura das barras no eixo Y
- width - espessura das barras
- color - cor

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



59

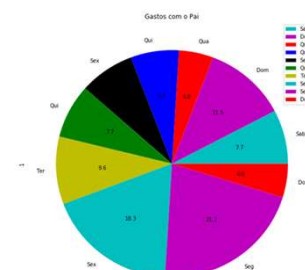
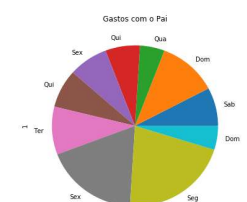


## plot.pie: gráfico de pizza

Principais atributos ajustáveis:

- legend - True/False
- autopct - formato – exibe percentual
- colors - lista de cores das partes

```
sGastos=pd.read_excel("gastosAlimPai.xlsx", header=None, index_col=0, squeeze=True,
                        decimal=',')
sGastos.plot.pie(title="Gastos com o Pai",
                  figsize=(6,6))
sGastos.plot.pie(title="Gastos com o Pai",
                  colors= ['c','m','r','b','k','g','y']
                  legend = True, autopct="%.1f",
                  figsize=(10,10))
```



60



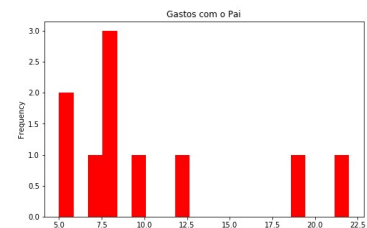
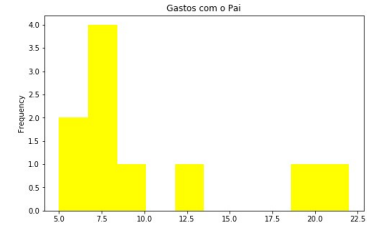
## plot.hist: histograma

Principais atributos ajustáveis:

- bins - quantidade de barras (bins)
- color - cor

Obs: o conjunto de valores deve ser válido senão retorna um erro

```
sGastos=pd.read_excel("gastosAlimPai.xlsx",header=None,index_col=0,squeeze=True,
                      decimal=',')
sGastos.plot.hist(title="Gastos com o Pai",
                  figsize=(8,5),
                  color='yellow')
sGastos.plot.hist(title="Gastos com o Pai",
                  figsize=(8,5),
                  color='r',
                  bins=20)
```



61



## Problema: Acrescentando Sumarizações

```
Seg: [ 35.28  9.23  9.39  8.84  8.58 40.28 ]
Ter: [ 12.74  8.32 40.89 55.27 12.74 ]
Qua: [ 65.18 37.35 55.19 48.32 13.84 32.33 42.40 ]
Qui: [ 65.23 41.52 31.91  9.39  8.51 31.91 ]
Sex: [ 41.58  3.64  8.54  5.34  8.80 61.63  5.34 ]
Sab: [ 76.80 56.16 61.50 61.29 45.79  9.38  6.33 ]
Dom: [ 71.22 69.32  9.35 17.89  9.70 ]
```

Para completar a análise dos gastos do Gasparzinho é necessário também apresentar um resumo dos dados.

Portanto, como exibir:

- Quantos gastos ocorreram?
- O total em cada dia? O gasto médio por dia?
- O total da semana? O gasto médio da semana?
- Em qual dia gastou mais?
- Em qual dia gastou menos?
- Qual o menor valor gasto? Em qual dia?
- Qual o maior valor gasto? Em qual dia?
- Exibir graficamente o total por dia

Sab	76,80
Dom	71,22
Qua	65,18
Qui	65,23
Sex	44,58
Qui	41,52
Ter	12,74
Sex	3,64

Qt: 43  
Média: 193.46  
Total: 1354.24  
...

62

# Descrição e Sumarização

63

## Descrição e Sumarização

### Medidas de Tendência Central ou Posição

**Média:**  
`series.mean()` <sup>[1]</sup>

```
>>>s.mean()
31.428571428571427

>>>s.mean(level=0)
a    28.333333
b    22.500000
c    45.000000
dtype: float64
```

**Mediana:**  
`series.median()` <sup>[1]</sup>

```
>>>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

64





## Descrição e Sumarização

### Medidas de Tendência Central ou Posição

Máximo: `series.max()` <sup>[1][2]</sup>  
 Mínimo: `series.min()` <sup>[1][2]</sup>  
 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
```

s:	q:
a 10	a 1
b 30	b 3
c 50	c 5
a 30	a 3
b 15	b 1
c 40	c 4
a 45	a 4

Quantil:  
`series.quantile (q=%)`  
 padrão q=0.5

```
>>>s.quantile()
30.0
print(s.quantile(0.9))
47.0
```

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

65



## Descrição e Sumarização

### Medidas de Dispersão

Amplitude: val max - val min

```
>>>s.max() - s.min()
40          #(50 - 10)
```

Variância:  
`series.var()` <sup>[1]</sup>

```
>>>s.var()
222.61904761904762
```

Desvio Padrão:  
`series.std()` <sup>[1]</sup>

```
>>>s.std()
14.920423841803142
```

Covariância:  
`series.cov(series)`

```
>>>s.cov(q)
22.5
```


Correlação:  
`series.corr(series)`

```
>>>s.corr(q)
0.98721777257162646
```

s:	q:
a 10	a 1
b 30	b 3
c 50	c 5
a 30	a 3
b 15	b 1
c 40	c 4
a 45	a 4

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

66



## Descrição e Sumarização


### Totalizações

---

<p><b>Soma:</b></p> <pre>series.sum() [1]</pre>	<pre>&gt;&gt;&gt;s.sum() 220  &gt;&gt;&gt;s.sum(level=0) a 85 b 45 c 90</pre>	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <table style="font-size: 0.8em;"> <tr><td><u>S:</u></td><td></td><td><u>G:</u></td><td></td></tr> <tr><td>a</td><td>10</td><td>a</td><td>1</td></tr> <tr><td>b</td><td>30</td><td>b</td><td>3</td></tr> <tr><td>c</td><td>50</td><td>c</td><td>5</td></tr> <tr><td>a</td><td>30</td><td>a</td><td>3</td></tr> <tr><td>b</td><td>15</td><td>b</td><td>1</td></tr> <tr><td>c</td><td>40</td><td>c</td><td>4</td></tr> <tr><td>a</td><td>45</td><td>a</td><td>4</td></tr> </table> </div>	<u>S:</u>		<u>G:</u>		a	10	a	1	b	30	b	3	c	50	c	5	a	30	a	3	b	15	b	1	c	40	c	4	a	45	a	4
<u>S:</u>			<u>G:</u>																															
a	10		a	1																														
b	30	b	3																															
c	50	c	5																															
a	30	a	3																															
b	15	b	1																															
c	40	c	4																															
a	45	a	4																															
<p><b>Quantidade:</b></p> <pre>series.count()</pre>	<pre>&gt;&gt;&gt;s.count() 7</pre>																																	
<p><b>Contagem de valores exclusivos:</b></p> <pre>series.value_counts() [2]</pre> <p>(Tabela de frequências)</p>	<pre>&gt;&gt;&gt;s.value_counts() 30 2 15 1 45 1 10 1 50 1 40 1 dtype: int64  &gt;&gt;&gt;s.index.value_counts() a 3 c 2 b 2 dtype: int64</pre>																																	

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

67



## Descrição e Sumarização

### Resumo

---

<p><b>Resumo:</b></p> <pre>series.describe()</pre>	<pre>&gt;&gt;&gt;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</pre>	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <table style="font-size: 0.8em;"> <tr><td><u>S:</u></td><td></td></tr> <tr><td>a</td><td>10</td></tr> <tr><td>b</td><td>30</td></tr> <tr><td>c</td><td>50</td></tr> <tr><td>a</td><td>30</td></tr> <tr><td>b</td><td>15</td></tr> <tr><td>c</td><td>40</td></tr> <tr><td>a</td><td>45</td></tr> </table> </div>	<u>S:</u>		a	10	b	30	c	50	a	30	b	15	c	40	a	45
<u>S:</u>																		
a	10																	
b	30																	
c	50																	
a	30																	
b	15																	
c	40																	
a	45																	

68



## Uma Solução: Acrescentando Sumarizações

```
def resumos(sG):
    print("\n\nResumos\n")
    print("\nQuantidade de gastos: ', sG.count())
    stot=sG.sum(level=0)
    print("\nTotal Gasto em cada dia: ', stot)
    print("\nGasto médio em cada dia: ', sG.mean(level=0))
    print("\nTotal Gasto na semana: ', sG.sum())
    print("\nGasto médio na semana: ', sG.mean())
    print("\nDia com Maior Gasto: {<s}'.format(stot.idxmax()), end='')
    print('\tValor: {:.2f}'.format(stot.max()))
    print("\nDia com Menor Gasto: {<s}'.format(stot.idxmin()), end='')
    print('\tValor: {:.2f}'.format(stot.min()))
    print("\n Maior Valor Gasto: {:.2f} - {s}'.format(sG.max(), sG.idxmax())
    print("\n Menor Valor Gasto: {:.2f} - {s}'.format(sG.min(), sG.idxmin())
    return
```

69



## Uma Solução: Acrescentando Sumarizações

""Para completar a análise dos gastos do Gasparzinho é necessário apresentar também um resumo dos dados. Portanto, como exibir: Quantos gastos ocorreram? O total em cada dia? O total da semana? O percentual de cada dia no gasto da semana? Em qual(is) dia(s) gastou mais? Qual o menor valor gasto? Em qual dia? Qual o maior valor gasto? Em qual dia?""

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

```
Seg: [ 35.28  9.23  9.39  8.84  8.58  40.28 ]
Ter: [ 12.74  8.32  40.89  55.27  12.74 ]
Qua: [ 65.18  37.35  55.19  48.32  13.84  32.33  42.40 ]
Qui: [ 65.23  41.52  31.91  9.39  8.51  31.91 ]
Sex: [ 41.58  3.64  8.54  5.34  8.80  61.63  5.34 ]
Sab: [ 76.80  56.16  61.50  61.29  45.79  9.38  6.33 ]
Dom: [ 71.22  69.32  9.35  17.89  9.70 ]
```

Resumos Gastos Gasparzinho

Quantidade de gastos: 43

Total Gasto em cada dia: 0

Dom 177.484

Qua 294.610

Qui 188.469

Sab 317.250

Seg 111.601

Sex 134.866

Ter 129.955

Name: 1, dtype: float64

Gasto médio em cada dia: 193.46

Total Gasto na semana: 1354.24

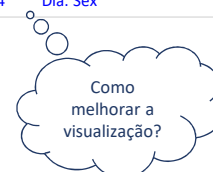
Gasto médio na semana: 31.49

Dia com Maior Gasto: Sab Valor: 317.25

Dia com Menor Gasto: Seg Valor: 111.60

Maior Valor Gasto: 76.80 Dia: Sab

Menor Valor Gasto: 3.64 Dia: Sex

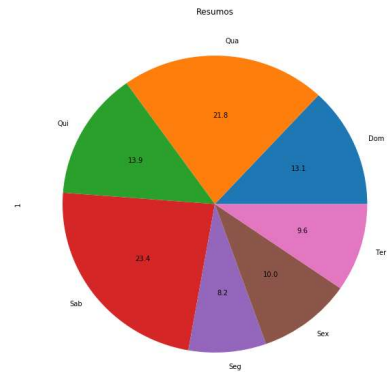


70

## Visualização Gráfica



```
sTot= sGastos.sum(level=0).
sTot.plot.bar(title="Resumos")
```



```
sTot= sGastos.sum(level=0).
sTot.plot.pie(title="Resumos",
autopct="%1f",figsize=(10,10))
```

71

## Manipulando Series

72



## Problema: Atualização dos dados da Series

Conferindo os valores, Gasparzinho observou que alguns gastos foram digitados errados. Todos os gastos da planilha entre R\$80,00 e R\$99,00 estavam incorretos, na verdade correspondiam a gastos entre R\$8,00 e R\$9,90. Todos os gastos com unidade 4 deveriam ter unidade 1. Por exemplo, o valor 92,32 que está na planilha é de um gasto de R\$9,23 e R\$34,91 deveria ser R\$31,91. Como consertá-los?

```

Seg: [ 35.28  9.23  9.38  8.84  8.57
      92.32  93.87  88.44  85.78  40.28 ]
Ter: [ 12.74  8.31  40.89  55.27  12.74 ]
Qua: [ 65.18  37.35  55.19  48.32  13.84  32.33  42.40 ]
Qui: [ 65.23  41.52  34.91  9.38  8.51
      8.54  8.79  93.86  85.13  34.91 ]
Sex: [ 44.58  3.64  85.40  5.34  87.96  61.63  5.34 ]
Sab: [ 76.80  56.16  64.50  64.29  45.79  93.80  6.33 ]
Dom: [ 71.22  69.32  93.54  17.89  97.03 ]

```

73



## 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

```

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

import math
s.apply(math.sqrt)
33A    6.324555
33B    3.872983
33C    4.242641
33E      NaN
33A    6.782330
dtype: float64

```

```

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

```

74



## Uma Solução: Consertando os valores incorretos

```
import pandas as pd
def corrige(x):
    if 80<=x<100:
        return x/10
    elif int(x)%10 == 4:
        return x-3
    else:
        return x
# Montar a Series
sGastos=pd.read_excel("gastosAlimMae.xlsx", header=None, index_col=0, squeeze=True,
                      decimal=',')
# Exibir dados por dia de semana
exibe(sGastos)
# Corrigir dados
sGastos=sGastos.apply(corrige)
```

```
Seg:[ 35.28  9.23  9.39  8.84  8.58 40.28 ]
Ter:[ 12.74  8.32 40.89 55.27 12.74 ]
Qua:[ 65.18 37.35 55.19 48.32 13.84 32.33 42.40 ]
Qui:[ 65.23 41.52 31.91 9.39 8.51 31.91 ]
Sex:[ 41.58 3.64 8.54 5.34 8.80 61.63 5.34 ]
Sab:[ 76.80 56.16 61.50 61.29 45.79 9.38 6.33 ]
Dom:[ 71.22 69.32 9.35 17.89 9.70 ]
```



75



## Problema: Acrescentando percentuais e sumarizações totais

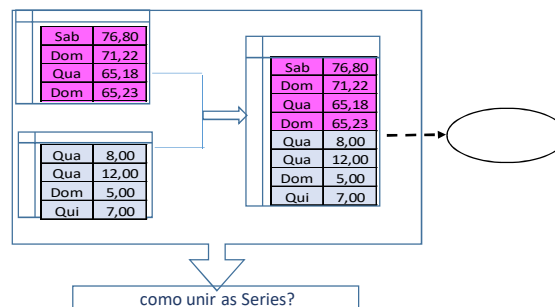
- I. Como saber o gasto percentual de cada dia?

$$\text{Percentual: } \frac{\sum \text{gastos do dia}_i}{\sum \text{gastos da semana}} * 100$$

como realizar operações aritméticas sobre a Series?

- II. E se houvesse outra planilha com os gastos anotados pelo pai do Gasparzinho?

Para os resumos gerais,  
estas Series devem ser unidas



76



## Análise do Problema: Calculando percentuais e usando 2 planilhas

### Idéia da Solução:

- I. Criar Series com gastos a partir do arquivo com as anotações da mãe, usando a 1ª coluna como índice
- II. Criar Series com gastos a partir do arquivo com as anotações do pai, usando a 1ª coluna como índice
- III. Exibir Series com os dados da mãe
- IV. Exibir Series com os dados do pai
- v. Sumarizar a Series com os dados da mãe
- vi. Sumarizar a Series com os dados do pai
- vii. Unir as Series do pai e da mãe
- viii. Sumarizar a Series com elementos do pai e da mãe

77



## Problema: Acrescentando percentuais e sumarizações totais

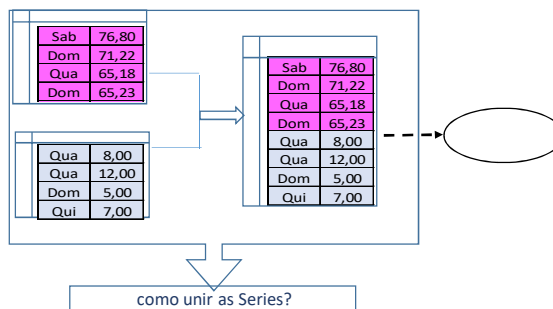
- I. Como saber o gasto percentual de cada dia?

$$\text{Percentual: } \frac{\Sigma \text{gastos do dia}_i}{\Sigma \text{gast da semana}} * 100$$

como realizar operações aritméticas sobre a Series?

- II. E se houvesse outra planilha com os gastos anotados pelo pai do Gasparzinho?

Para os resumos gerais,  
estas Series devem ser unidas



78



## Análise do Problema: Calculando percentuais e usando 2 planilhas

### Idéia da Solução:

- I. Criar Series com gastos a partir do arquivo com as anotações da mãe, usando a 1ª coluna como índice
- II. Criar Series com gastos a partir do arquivo com as anotações do pai, usando a 1ª coluna como índice
- III. Exibir Series com os dados da mãe
- IV. Exibir Series com os dados do pai
- V. Sumarizar a Series com os dados da mãe
- VI. Sumarizar a Series com os dados do pai
- VII. Unir as Series do pai e da mãe
- VIII. Sumarizar a Series com elementos do pai e da mãe

79



## Uma Solução: Acrescentando percentuais e sumarizações totais

```
def resumos(sG):
    print("\n\nResumos\n")
    print("\nQuantidade de gastos: ', sG.count())
    sTot=sG.sum(level=0)
    tSem= sG.sum()
    print('\nTotal Gasto em cada dia: ', sTot)
    print('\nGasto médio em cada dia: ', sG.mean(level=0))
    print('\nTotal Gasto na semana: ', sG.sum())
    print('\nGasto médio na semana: ', sG.mean())
    print('\nDia com Maior Gasto: {<s}'.format(sTot.idxmax()), end="")
    print('\tValor: {:.2f}'.format(sTot.max()))
    print('\nDia com Menor Gasto: {<s}'.format(sTot.idxmin()), end="")
    print('\tValor: {:.2f}'.format(sTot.min()))
    print('\nMaior Valor Gasto: {:.2f} - {s}'.format(sG.max(), sG.idxmax()))
    print('\nMenor Valor Gasto: {:.2f} - {s}'.format(sG.min(), sG.idxmin()))
    print('\n Percentuais Diários: ', sTot/tSem)
    return
```

80





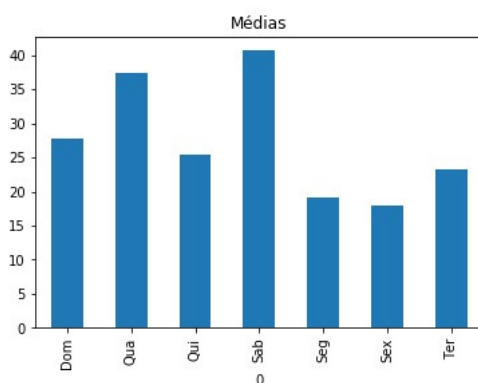
## Uma Solução: Acrescentando percentuais e sumarizações totais

```
sMae=pd.read_excel("gastosAlimMaeCor.xlsx", header=None,index_col=0, squeeze=True,
                  decimal=',')
sMae=pd.read_excel("gastosAlimPai.xlsx", header=None,index_col=0, squeeze=True, decimal=',')
exibe("MÃE", sMae)
resumos("MÃE",sMae)
exibe("PAI", sPai)
resumos("PAI",sPai)
sGeral=sMae.append(sPai)
exibe("GERAL",sGeral)
resumos("GERAL",sGeral)
```

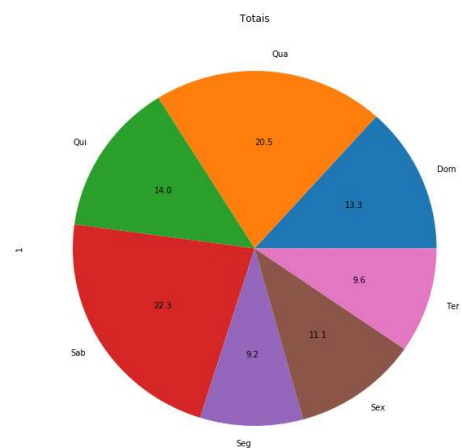
81



## Visualização Gráfica



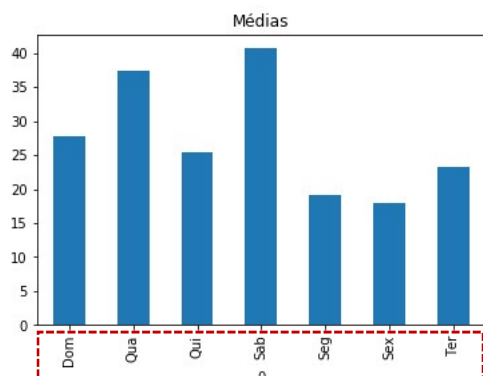
```
sTot= sGastos.mean(level=0).
sTot.plot.bar(title="Médias")
```



```
sTot= sGeral.sum(level=0).
sTot.plot.pie(title="Totais",
              autopct="%.1f",figsize=(10,10))
```

82

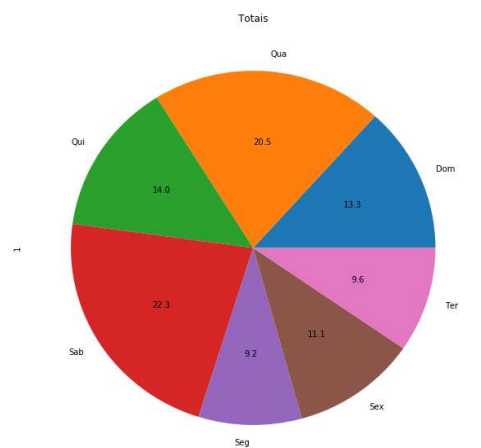
## Visualização Gráfica



```
sTot= sGastos.mean(level=0).  
sTot.plot.bar(title="Médias")
```



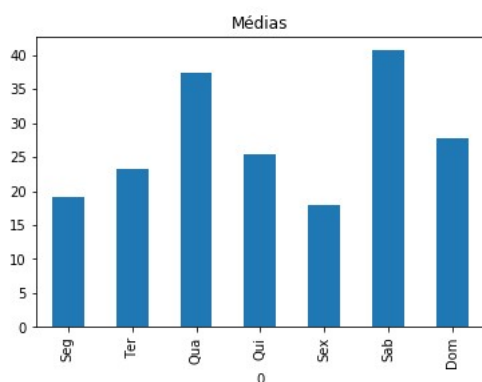
Os métodos de sumarização ordenam os índices→  
Reindexar a *series* resultante: método `reindex(novoíndice)`



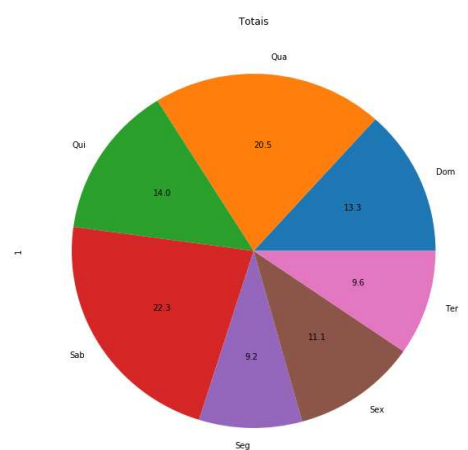
```
sTot= sGeral.sum(level=0).  
sTot.plot.pie(title="Totais",  
autopct="%.1f",figsize=(10,10))
```

83

## Visualização Gráfica



```
sTot= sGastos.mean(level=0).  
sTot = sTot.reindex(['Seg','Ter','Qua','Qui','Sex','Sab','Dom'])  
sTot.plot.bar(title="Médias")
```



```
sTot= sGeral.sum(level=0).  
sTot.plot.pie(title="Totais",  
autopct="%.1f",figsize=(10,10))
```

84

# Alterando Series

85

## Unir, Substituir e Atualizar Series

**Sintaxe:** `series.append(series)` - Cria uma cópia com os elementos da Series recebida incluídos no final

`series.replace(to_replace=valor, value=novo) *` - Cria uma cópia, substituindo todas as ocorrências de valor por novo

`series.update(series)` - Altera atuais valores pelos valores da Series recebida, alinhando pelo índice

\* com `inplace=True`, realiza a operação na Series, não cria uma cópia

**s1:**  
Frios 1.5  
Leite 1.0  
Pão 10.0

**s2:**  
Frios 1.5  
Leite 4.0  
Chá 1.5  
Pão 3.0

**s3:**  
Cereal 3.0  
Ovo 2.0  
Pão 10.0  
Frios 90.0

```
>>>s1.update(s2)
Frios 1.5
Leite 4.0
Pão 3.0
```

```
>>>sC=s1.append(s2)
>>>sC
Frios 1.5
Leite 1.0
Pão 10.0
Frios 1.5
Leite 4.0
Chá 1.5
Pão 3.0
```

```
>>>sC.replace(to_replace=1.5,value=99)
Frios 99.0
Leite 1.0
Pão 10.0
Frios 99.0
Leite 4.0
Chá 99.0
Pão 3.0
```

86

## Mãos na Massa

Diariamente, 3 agentes da Vigilância Sanitária visitam uma região para detectar focos de larvas do mosquito *Aedes aegypti* nos seguintes locais:

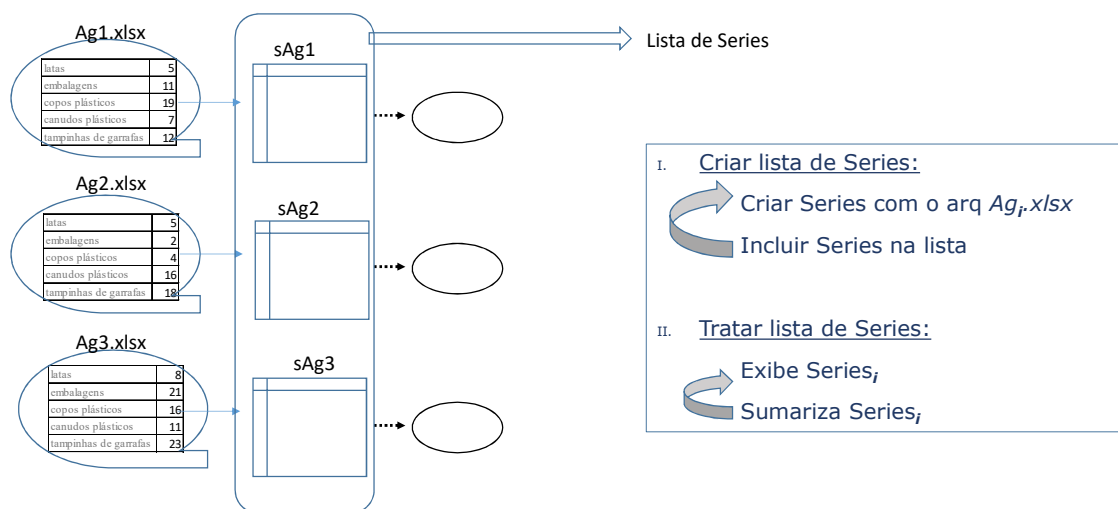
latas, embalagens, copos e canudos plásticos, garrafas, tampinhas de garrafas, vasos de plantas, jarros de flores, bromélias, caixas d'água, tambores, latões, cisternas, calhas, piscinas, vasos sanitários, pneus velhos, sacos plásticos, lixeiras, bueiros, ralos, lonas e lajes.

A quantidade de focos encontrados por localização são registradas em arquivos Excel denominados  $Ag_n$ , onde  $n$  varia de 1 a 3. Caso não seja encontrado focos em alguma localização, ela não consta no arquivo.

Construa um script que mostre para cada agente, a quantidade de focos encontrada por tipo de localização, a localização com maior incidência, quantidade total e a distribuição gráfica das quantidades por localização.

87

## Análise do Problema: Focos por Agente



88



## Uma Solução: Focos por Agente

```
import pandas as pd

def geraNome(n):
    return "AG"+str(n+1)

def resumos(nome,sG):
    print("\nResumos {}".format(nome))
    print('Total de focos:', s.sum())
    print('Local de maior incidência: {} com {} focos'.format(s.idxmax(),s.max()))
    return

def carregarSeries():
    #Cria o nome do arquivo do agente
    lSeries=[]
    for i in range(0,3):
        #Monta o nome do arquivo do agente
        arq=geraNome(i)+".xlsx"
        #Cria a Series a partir do arquivo
        s=pd.read_excel(arq,header=None,squeeze=True,index_col=0)
        #Adiciona a Series à lista
        lSeries.append(s)
    return lSeries
```

89



## Uma Solução: Focos por Agente

```
#Cria lista de Series com dados de cada agente
lSeries=carregarSeries()
#Sumarização por agente
for i,s in enumerate(lSeries):
    nome=geraNome(i)
    print("\n{}".format(nome))
    print(s)
    resumos(nome,s)
```

AG1	
latas	5
embalagens	11
copos plásticos	19
tampinhas de garrafas	12
vasos de plantas	12
jarros de flores	15
caixas d'água	15
tambores	15
latões	24
cisternas	23
calhas	8
vasos sanitários	8
sacos plásticos	12
lixeiras	23
bueiros	17
ralos	20
lonas	13
lajes	21

Resumos AG1  
Total de focos: 273  
Local de maior incidência: latões  
com 24 focos

AG2	
embalagens	2
copos plásticos	4
canudos plásticos	16
tampinhas de garrafas	18
vasos de plantas	10
jarros de flores	25
bromélias	8
tambores	25
latões	9
cisternas	4
calhas	11
piscinas	22
sacos plásticos	24
lixeiras	13
bueiros	7
ralos	15
lonas	23

Resumos AG2  
Total de focos: 236  
Local de maior incidência: jarros  
de flores com 25 focos

AG3	
latas	8
copos plásticos	16
canudos plásticos	11
tampinhas de garrafas	23
vasos de plantas	16
jarros de flores	23
bromélias	28
caixas d'água	19
latões	5
cisternas	1
calhas	22
piscinas	3
vasos sanitários	24
sacos plásticos	7
lixeiras	10
ralos	18
lonas	5
lajes	4

Resumos AG3  
Total de focos: 243  
Local de maior incidência:  
bromélias com 28 focos

90

## Análise da Solução: Totais Gerais

Como saber

- I. O total de focos encontrados na região por localização?
 

Somar por localização as quantidades encontradas por cada agente: somar as Series por localização de cada agente
- II. O total de focos na região?
 

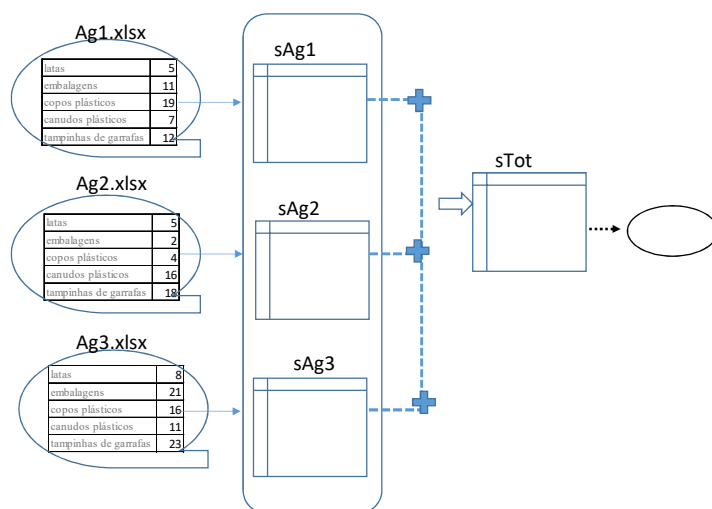
Somar as quantidades encontradas por cada agente (somar as Series de cada agente)
- III. Qual a localização com mais focos na região?
 

Encontrar o maior valor após a soma das quantidades encontradas por cada agente
- IV. A distribuição gráfica dos focos por localização na região?
 

Construir um gráfico de pizza com o total de focos encontrados por região

91

## Análise da Solução: Totais Gerais



- I. Somar Series → sTot
- II. Exibir sTot
- III. Sumarizar sTot

92

# Operações com Series

93

## Operações Aritméticas de Series

com operadores

**Sintaxe:**`series operador series`

Objetos de Series semelhantes podem ser combinados com operações aritméticas. Os dados são alinhados pelo índice. Retorna a Series resultante da operação. Se o índice € às duas Series, os valores são operados, nos demais casos retorna *Null/NaN*.

**s1:**

Frios	0.5
Leite	1.0
Pão	10.0

**s2:**

Frios	1.0
Leite	3.0
Pão	20.0



**s3:**

Cereal	3
Ovo	2
Pão	10

```
>>>s2/s1
Frios    2.0
Leite    3.0
Pão      2.0
dtype: float64
```

```
>>>s1 - s2
Frios   -0.5
Leite   -2.0
Pão    -10.0
dtype: float64
```

```
>>>s2 + s3
Cereal    NaN
Frios     NaN
Leite     NaN
Ovo       NaN
Pão      30.0
dtype: float64
```

94



## Operações Aritméticas de Series

com métodos

### Sintaxe:

```
series.metodoOperação(series, fill_value=valor)
```

Objetos de Series semelhantes podem ser combinados com métodos que implementam as operações aritméticas. Os dados são alinhados pelo índice. Retorna a Series resultante da operação. Se o argumento `fill_value` está presente, quando não há sobreposição nos índices, utiliza `valor` para o cálculo. Quando ambas Series tem NaN no índice, retorna NaN.

```
s1:
Frios    0.5
Leite     1.0
Pão      10.0
```

```
s2:
Frios    1.0
Leite     3.0
Pão      20.0
```

```
s3:
Cereal    3
Ovo        2
Pão       10
```

```
>>>s2.add(s3)
Cereal    NaN
Frios     NaN
Leite     NaN
Ovo       NaN
Pão       30.0
dtype: float64
```

```
>>>s2.mul(s1)
Frios     0.5
Leite     3.0
Pão      200.0
dtype: float64
```

```
>>>s2.add(s3, fill_value=0)
Cereal     3.0
Frios      1.0
Leite      3.0
Ovo        2.0
Pão       30.0
dtype: float64
```

95



## Operações Aritméticas de Series

com índices duplicados

```
s1:
Frios    0.5
Leite     1.0
Pão      10.0
Frios     2.5
```

```
s2:
Frios    1.0
Leite     3.0
Frios    20.0
Leite     4.0
```

```
s3:
Cereal    3
Ovo        2
Pão       10
Pão        3
```

```
>>>s1.add(s2, fill_value=0)
Frios      1.5      #0.5 + 1.0
Frios     20.5      #0.5 + 20.0
Frios       3.5      #2.5 + 1.0
Frios     22.5      #2.5 + 20.0
Leite       4.0
Leite       5.0
Pão        10.0
dtype: float64
```

```
>>>s1.add(s3, fill_value=0)
Cereal      3.0
Frios       0.5
Frios       2.5
Leite       1.0
Ovo         2.0
Pão        20.0
Pão        13.0
dtype: float64
```

96





## Uma Solução: Somando as Series

```
import pandas as pd

def trataSeriesIndiv():
    #Cria lista de Series com dados de cada agente
    lSeries=carregarSeries(3)
    #Sumarização por agente
    for (i,s) in enumerate(lSeries):
        nome=geraNome(i)
        print("\n{}".format(nome))
        print(s)
        resumos(nome,s)
    return lSeries

def somaSeries(l):
    sTot=l[0].copy()
    for i in range(1, len(l)):
        sTot=sTot.add(l[i], fill_value=0)
    return sTot
```

97

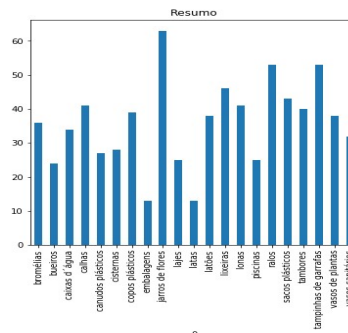


## Uma Solução: Somando as Series

```
#Exibe e sumariza cada Series
lSeries=trataSeriesIndiv()
#Soma as Series
sTot = somaSeries(lSeries)
#Exibe e sumariza Totais
print(sTot)
#Visualiza Graficamente Totais
sTot.plot.pie(title="Resumo", figsize=(6,6))
```

TOTAIS	
bromélias	36.0
bueiros	24.0
caixas d'água	34.0
calhas	41.0
canudos plásticos	27.0
cisternas	28.0
copos plásticos	39.0
embalagens	13.0
jarros de flores	63.0
lajes	25.0
latas	13.0
latões	38.0
lixeiras	46.0
lonas	41.0
piscinas	25.0
ralos	53.0
sacos plásticos	43.0
tambores	40.0
tampinhas de garrafas	53.0
vasos de plantas	38.0
vasos sanitários	32.0

Resumos TOTAIS  
 Total de focos: 752.0  
 Local de maior incidência: jarros  
 de flores com 63.0 focos



98



## Mãos na massa

Considerando que foi aberta uma nova turma do currículo atual, 33D, e no 2º dia de matrícula foram realizadas as seguintes inscrições:

dicionário:={'33A':3,'33C':2,'33F':4,'33B':5,'33E':2,'A44':2, 'A33':2, '33D':12}

Mostrar a ocupação de cada turma e a quantidade de vagas por turma.

*Lembre-se que as turmas do currículo novo tem 50 vagas e do antigo, 40.*

```
dInscD1={'33A':40, '33C':40, '33F':16, '33B':15, '33E':None, 'A44':None, 'A33':25}
sInscD1=pd.Series(dInscD1)
```

Inscrições 1º dia:

```
33A 40.0
33B 15.0
33C 40.0
33E NaN
33F 16.0
A33 25.0
A44 NaN
dtype: float64
```

99



## Mãos na massa

### Idéia da Solução:

- I. Criar Series do 1º dia de matrícula
- II. Criar Series do 2º dia de matrícula
- III. Criar Series do total de vagas
  1. Criar com valor 50 e *index* de uma das series anteriores
  2. Substituir valores das turmas do currículo antigo por 40
- IV. Calcular Ocupação atual (soma das Series por dia de matrícula)
 

Problemas na Series do 1º dia:  
NaN e falta da turma D
- V. Calcular Oferta atual (total de Vagas - Ocupação total)
- VI. Exibir Series

1º Dia		2º Dia	
33A	40.0	33A	3.0
33B	15.0	33B	5.0
33C	40.0	33C	2.0
33E	NaN	33D	12.0
33F	16.0	33E	2.0
A33	25.0	33F	4.0
A44	NaN	A33	2.0
		A44	2.0

Total de Vagas	
33A	50
33B	50
33C	50
33D	50
33E	50
33F	50
A33	40
A44	40

Soma Inscrições	
33A	43.0
33B	20.0
33C	42.0
33D	12.0
33E	2.0
33F	20.0
A33	2.0
A44	2.0

100



## Uma Solução com operador

```
import pandas as pd
dInscD1={'33A':40,'33C':40,'33F':16,'33B':15,'33E':None,'A44':None, 'A33':25}

sInscD1=pd.Series(dInscD1)
sInscD2=pd.Series({'33A':3,'33C':2,'33F':4,'33B':5,'33E':2,'A44':2, 'A33':2, '33D':12})

# Arrumar a series do 1º dia p/conter mesmos índices da series do 2º dia
sInscD1['33D']=0 #inclui a nova turma na series do 1º dia
sInscD1[['33E','A44']]=0 # substitui os valores ausentes

sTot=pd.Series(50,sInscD2.index) #cria series com vagas totais
sTot[['A44','A33']]=40 #altera quantidades das turmas antigas

sOc=sInscD1+sInscD2
sLiv = sTot-sOc
print(sInscD1)
print(sInscD2)
print(sOc)
print(sLiv)
```

Por método:  
series.fillna(valor):  
sInscD1.fillna(0)

101



## Outra Solução com método

```
import pandas as pd
dInscD1={'33A':40,'33C':40,'33F':16,'33B':15,'33E':None,'A44':None, 'A33':25}

sInscD1=pd.Series(dInscD1)

sInscD2=pd.Series({'33A':3,'33C':2,'33F':4,'33B':5,'33E':2,'A44':2, 'A33':2,
                  '33D':12})

sTot=pd.Series(50,sInscD2.index) #cria series com vagas totais
sTot[['A44','A33']]=40 #altera quantidades das turmas antigas

sOc=sInscD1.add(sInscD2, fill_value=0)
sLiv = sTot-sOc
print(sInscD1)
print(sInscD2)
print(sOc)
print(sLiv)
```

102

# Filtros

Seleção de itens que satisfazem um critério

103

## Problema: Acrescentando Filtros

- I. Quais dos valores gastos pelo Gasparzinho (e seus dias) foram superiores a R\$25,00?

*Mostrar valor e índice apenas  
dos itens que satisfazem o critério*

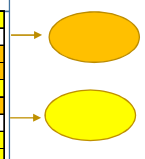
Sab	76,80
Dom	1,10
Seg	6,15
Ter	7,00
Sex	44,58
Sab	1,52
Dom	12,74
Sex	25,64
Seg	35,28

- II. Qual o percentual ou média dos valores superiores a R\$25,00?

- III. Quantos valores gastos foram inferiores a R\$ 10,00

*Sumarizar apenas os itens  
que satisfazem o critério*

Sab	76,80
Dom	1,10
Qua	6,15
Qui	7,00
Sex	44,58
Qui	1,52
Sex	12,74
Sex	25,64
Seg	35,28



104



## Pertinência no Index: in

### Sintaxe:

`valor in series`

Retorna True se valor é um índice da Series ou False caso contrário.

```
'33E' in vagas
True
```

```
'33R' in vagas
False
```

105



## Seleção condicional de itens da Series

### Sintaxe:

`series operador_lógico condição`

Retorna uma nova Series de valores booleanos True/False.

- Pode-se usar a Series de booleanos para filtrar os itens selecionados (com valor True).

Conectivos Lógicos: e: & ou: | não: ~

```
sC=pd.Series({'Pão':10,'Leite':2,'Ovo': 1})
>>>sC
Leite    2
Ovo      1
Pão     10
dtype: float64

>>>s=sC < 5
>>>s
Leite    True
Ovo     True
Pão     False
dtype: bool
```

```
>>>sC.loc[s]
# [V,V,F]
Leite    2
Ovo      1
dtype: int64

>>> sC.loc[sC<5]
Leite    2
Ovo      1
dtype: int64
```

106



## Filtro: método .isnull

**Sintaxe:** `series.isnull()`  
`series.notnull()`

**isnull:** retorna uma nova *Series* com mesmos índices e `True` onde o valor está ausente (Null/NaN) e `False`, em caso contrário.

**notnull:** `True` quando o valor não está ausente e `False`, caso contrário

**sN**  
 Chá NaN  
 Ovo 1.0  
 Pão 10.0  
 Uva NaN

```
>>>sN.isnull()
Chá    True
Ovo    False
Pão    False
Uva    True
dtype: bool

>>>sN.notnull()
Chá    False
Ovo    True
Pão    True
Uva    False
dtype: bool
```

```
>>> sN.loc[sN.notnull()] # [F,V,V,F]
Ovo    1.0
Pão    10.0
dtype: float64

>>>sN.loc[sN.isnull()] # [V,F,F,V]
Chá    NaN
Uva    NaN
dtype: float64
```

107



## Filtro: método .isin

**Sintaxe:** `series.isin(lista de valores)`  
`series.index.isin(lista de índices)`

Retorna uma *Series* com `True` onde o elemento da *Series* é lista de valores. Útil para selecionar linhas com os valores desejados.

Para o *Index* retorna um vetor booleano. Útil quando não se sabe quais dos rótulos procurados estão de fato presentes.

**sC:**  
 Leite 2  
 Ovo 1  
 Pão 10

```
sC.isin([4,10])
Leite   False
Ovo     False
Pão     True

s=sC.index.isin(['Pão','j',0])
array([False, False,  True],
      dtype=bool)
```

```
sC.loc[['Pão','j',0]]
Pão    10.0
j       NaN
0       NaN
dtype: float64
```



NaN/Null em índices ≠

valores apenas onde é True

```
sC.loc[s]
Pão    10
dtype: int64
```

# [F,F,V]

108



## Filtro de strings

*Series* e *Index* possuem um conjunto de métodos de processamento de strings que operam sobre seus elementos. São acessados através do atributo **str**. Em geral, têm o mesmo nome de seus equivalentes no tipo string.

Sintaxe:

```
series.str.método()
series.str.propriedade
```

```
series.str.contains(padrão)
series.str.index.contains(padrão)
```

```
series.str.match(padrão)
series.str.index.match(padrão)
```

```
k=pd.Series(['ana','bet','car','kor','tor','mol','bal'], ['lb','bb','c','db','af','fa','br'])
```

```
>>>k.loc[k.str.contains('a')]
ab    ana
c     car
br    bal
dtype: object

# tem 'a'

>>>k.loc[k.index.str.contains('a')]
af    tor
fa    mol
dtype: object
```

```
>>>k.loc[k.str.match('a')]
lb    ana
dtype: object

# iniciam com 'a'

>>>k.loc[k.index.str.match('a')]
af    tor
dtype: object

>>>k[k.index.str.match('r')]
Series([], dtype: object)
```

109



## Alteração com Filtro

Sintaxe:

```
series.loc[critério]= valor ou lista de valores
```

critério: produz uma *Series/array* booleano

Altera o valor(es) do(s) elemento(s) indexado(s) por posições onde o valor é True

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

>>>s.loc[s<20]=30
>>>s
33A    40.0
33B    30.0
33C    30.0
33E     NaN
33A    46.0
dtype: float64

# FVVFF

>>>sIx=s.index.str.contains('3A')
array([ True, False, False, False,  True],
      dtype=bool)

>>>s.loc[sIx]=70
>>>s
33A    70.0
33B    30.0
33C    30.0
33E     NaN
33F    70.0
dtype: float64

# VFFFF
```

```
>>>s.loc[s.isin([46,70])]=20
>>>s
33A    20.0
33B    30.0
33C    30.0
33E     NaN
33F    20.0
dtype: float64

# VFFFV

>>>s.loc[s.isnull()]=0
>>>s
33A    20.0
33B    30.0
33C    30.0
33E     0.0
33F    20.0
dtype: float64

# FFFVF
```

Por método:  
series.fillna(val)  
s.fillna(0)

110



## Descartar elementos com Filtro

### Sintaxe:

```
series.drop(series[critério]) *
```

**critério:** produz uma *Series/array* booleano

Retorna uma cópia da *series* sem os elementos cujo valor na *Series* é False

\* **inplace=True**, altera o original  
**.drop**, espera *labels* de índices

**s:**

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

```
>>>f=s.index.str.contains('A')
# [True, False, False, False, True]
```

```
>>>s.drop(s.index[f])
# s.index[f]: ['33A', '33A']
```

```
33B 15.0
33C 18.0
33E NaN
dtype: float64
```

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

```
>>>s.drop([s.index[s>20]])
# s>20: V F F F V
# s.index[s>20]: ['33A', '33A']
```

```
33B 15.0
33C 18.0
33E NaN
dtype: float64
```

111



## Uma Solução: Acrescentando Filtros

```
def resumosFiltros(sG):
    # Series com os valores gastos superiores a R$25,00
    s25=sG[sG>25]
    print("\nGastos > R$25 e dias em que ocorreram", s25)
    # Percentual de Gastos superiores a R$25
    percG=s25.sum()/sG.sum()*100
    print("\nPercentual de Gastos > R$25", percG)
    # Distribuição do Percentual Geral de Gastos > R$25
    percDS=s25.sum(level=0)/sG.sum()*100
    print("\nPercentual de Gastos > R$25 por dia da Semana em relação ao total", percDS)
    # Distribuição do Percentual no Conjunto de Gastos > R$25
    percDS25=s25.sum(level=0)/s25.sum()*100
    print("\nPercentual de Gastos > R$25 por dia da Semana dentro do conjunto", percDS25)
    # Quantidade de valores inferiores a R$10,00
    qt10=sG[sG<10].count()
    print(sG[sG<10])
    print("\nQuantidade de gastos < R$10", qt10)
    return
```

112





## Uma Solução: Acrescentando Filtros

```
'''Quais dos valores gastos (e seus dias) foram superiores a R$25,00? Qual o
percentual de valores > R$25,00 em relação ao total? Qual a distribuição deste
percentual pelos dias da semana? E dentro do próprio conjunto? Quantos valores
gastos foram < a R$ 10,00'''

sMae=pd.read_excel("gastosAlimMae.xlsx", header=None, index_col=0,squeeze=True,
decimal=',')
sPai=pd.read_excel("gastosAlimPai.xlsx", header=None, index_col=0,squeeze=True, decimal=',')
geral=sMae.append(sPai)
resumosFiltros(geral)
```

113



## Mãos na Massa

Criar uma cópia (sCop) com os formatos de arquivos da *series* sF1\_l com área de armazenamento entre 500Kb e 2000Kb

```
>>>sF1_l[(sF1_l>500) & (sF1_l<2000)]
1    1118
2     665
dtype: int64
```

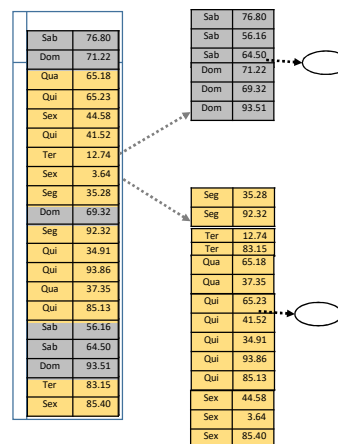
**Alternativa:** usar o método `series.between(esq, dir, inclusive=True/False)`:

```
sF1_l[sF1_l.between(500,2000,inclusive=False)]
```

114

## Problema: Agrupamentos

- I. Como agrupar os gastos por dia para exibi-los sem precisar do dicionário?
- II. Como agrupar os gastos dos dias úteis (segunda à sexta) e do final de semana e exibi-los sem precisar do dicionário?
- III. Qual o total de gastos nos dias úteis e o total no final de semana? Qual a relação percentual entre eles? Qual o gasto médio nos dias úteis e no final de semana?
- IV. Quantos gastos ocorreram por dia?



Sab	76.80
Dom	71.22
Qua	65.18
Qui	65.23
Sex	44.58
Qui	41.52
Ter	12.74
Sex	3.64
Seg	35.28
Dom	69.32
Seg	92.32
Qui	34.91
Qui	93.86
Qua	37.35
Qui	85.13
Sab	56.16
Sab	64.50
Dom	93.51
Ter	83.15
Sex	85.40

Sab	76.80
Sab	56.16
Sab	64.50
Dom	71.22
Dom	69.32
Dom	93.51

Seg	35.28
Seg	92.32
Ter	12.74
Ter	83.15
Qua	65.18
Qua	37.35
Qui	65.23
Qui	41.52
Qui	34.91
Qui	93.86
Qui	85.13
Sex	44.58
Sex	3.64
Sex	85.40

115

## Agrupamentos: dividir, aplicar e combinar

116



## Agrupamentos "Group by"

Processo que envolve um ou mais dos seguintes passos:

- ✓ Dividir os dados em grupos com base em alguns critérios
- ✓ Aplicar uma função a cada grupo de forma independente:
  - ✓ Agregação: computação de resumo (ou estatísticas) sobre cada grupo, como por exemplo:
    - soma ou média dos grupos
    - Tamanho / contagens nos grupos
  - ✓ Transformação: execução de operações específicas ao grupo, retornando uma estrutura similar, como por exemplo:
    - Padronizar dados dentro do grupo
    - Preencher valores ausentes com um valor derivado do grupo
  - ✓ Filtro: descartar valores a partir de computações que retornam V/F, como por exemplo:
    - Descartar dados usando a soma ou a média do grupo
    - Descartar dados que pertençam a grupo com poucos elementos
- ✓ Combinar os resultados em uma estrutura de dados

117



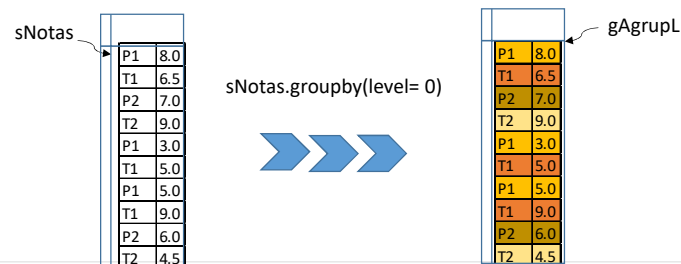
## Agrupando Series pelo índice

Sintaxe:

```
series.groupby(level=0)
```

Agrupar elementos com mesmo índice, atribuindo o valor do índice ao nome do grupo  
Retorna um objeto GroupBy

```
import pandas as pd
sNotas = pd.read_excel("notas.xlsx", header=None, index_col=0, squeeze=True)
gAgrupL = sNotas.groupby(level=0)
```



118

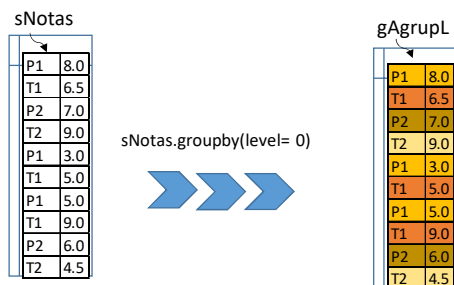


## Agrupando Series pelo índice

**Sintaxe:** `GroupBy.groups`  
`GroupBy.indices`

`GroupBy.groups` dict {nome do grupo -> labels dos índices do grupo}  
`GroupBy.indices` dict {nome do grupo -> array com a posição dos índices do grupo}

`gAgrupL.groups`  
`gAgrupL.indices`



groups:

```
{'P1': Index(['P1', 'P1', 'P1'], dtype='object', name=0),
 'P2': Index(['P2', 'P2'], dtype='object', name=0),
 'T1': Index(['T1', 'T1', 'T1'], dtype='object', name=0),
 'T2': Index(['T2', 'T2'], dtype='object', name=0)}
```

indices:

```
{'P1': array([0, 4, 6], dtype=int64),
 'P2': array([2, 8], dtype=int64),
 'T1': array([1, 5, 7], dtype=int64),
 'T2': array([3, 9], dtype=int64)}
```

119



## Agrupando Series por função

**Sintaxe:** `series.groupby(by=função)`

Agrupar elementos com mesmo índice, atribuindo o valor do índice ao nome do grupo  
 Retorna um objeto GroupBy

```
def fTipo(x):
    if 'P' in x:
        return 'Prova'
    else:
        return 'Teste'

sNotas = pd.read_excel("notas.xlsx", header=None, index_col=0, squeeze=True)
gAgrupF = sNotas.groupby(by=fTipo)
```



120

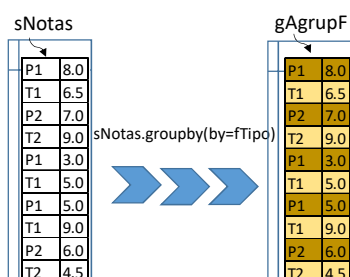


## Agrupando Series por função

**Sintaxe:** `GroupBy.groups`  
`GroupBy.indices`

`GroupBy.groups` dict {nome do grupo -> labels dos índices do grupo}  
`GroupBy.indices` dict {nome do grupo -> array com a posição dos índices do grupo}

`gAgrupF.groups`  
`gAgrupF.indices`



**groups:**  
 {'Prova': Index(['P1', 'P2', 'P1', 'P1', 'P2'], dtype='object', name=0),  
 'Teste': Index(['T1', 'T2', 'T1', 'T1', 'T2'], dtype='object', name=0)}

**indices:**  
 {'Prova': array([0, 2, 4, 6, 8], dtype=int64),  
 'Teste': array([1, 3, 5, 7, 9], dtype=int64)}

121

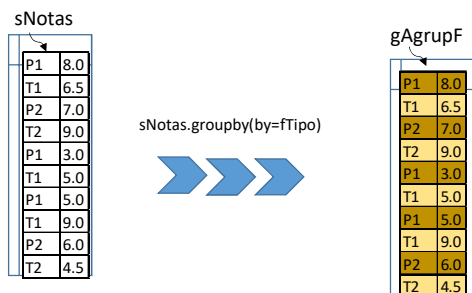


## Criando Series a partir de grupo

**Sintaxe:** `GroupBy.get_group(nome)`

Constrói uma Series com os elementos do grupo cujo nome foi fornecido

`gAgrupF.get_group('Prova')`



`gAgrupF.get_group('Prova')`

P1 8.0  
 P2 7.0  
 P1 3.0  
 P1 5.0  
 P2 6.0  
 Name: 1, dtype: float64

122

## Tamanho de um grupo

Sintaxe:

```
GroupBy.size()
```

Retorna a quantidade de elementos de cada grupo

```
gAgrupF.size()
```

sNotas

P1	8.0
T1	6.5
P2	7.0
T2	9.0
P1	3.0
T1	5.0
P1	5.0
T1	9.0
P2	6.0
T2	4.5

sNotas.groupby(by=fTipo)



gAgrupF

P1	8.0
T1	6.5
P2	7.0
T2	9.0
P1	3.0
T1	5.0
P1	5.0
T1	9.0
P2	6.0
T2	4.5

sNotas.groupby.size()



Prova 5  
Teste 5  
Name: 1, dtype: int64

123

## Problema: Exibindo grupos separadamente

- I. Como agrupar os gastos por dia para exibi-los sem precisar do dicionário?
- II. Como agrupar os gastos dos dias úteis (segunda à sexta) e do final de semana e exibi-los sem precisar do dicionário?

Sab	76.80
Dom	71.22
Qua	65.18
Qui	65.23
Sex	44.58
Qui	41.52
Ter	12.74
Sex	3.64
Seg	35.28
Dom	69.32
Seg	92.32
Qui	34.91
Qui	93.86
Qua	37.35
Qui	85.13
Sab	56.16
Sab	64.50
Dom	93.51
Ter	83.15
Sex	85.40

Sab	76.80
Sab	56.16
Sab	64.50
Dom	71.22
Dom	69.32
Dom	93.51

Seg	35.28
Seg	92.32
Ter	12.74
Ter	83.15
Qua	65.18
Qua	37.35
Qui	65.23
Qui	41.52
Qui	34.91
Qui	93.86
Qui	85.13
Sex	44.58
Sex	3.64
Sex	85.40

124

# Agregação

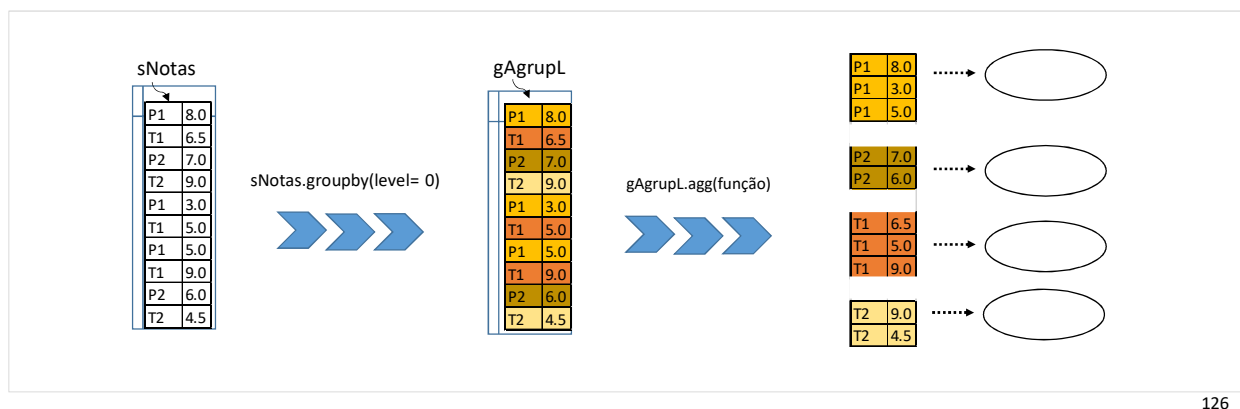
125

## Funções de Agregação

**Sintaxe:** `GroupBy.agg (função pré-definida ou nome de método )`  
`GroupBy.agg (lista de funções/nome de método)`

As funções fornecidas para agregação reduzem a dimensão do objeto fornecido. São aplicadas sobre os valores do grupo e retornam um resultado para o conjunto.

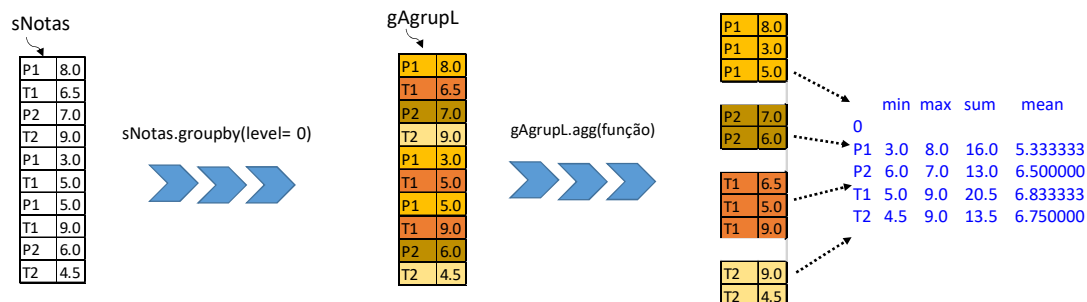
As mais comuns são *mean*, *sum*, *size*, *count*, *std*, *var*, *sem*, *describe*, *first*, *last*, *nth*, *min*, *max*.



126

## Exemplo Funções de Agregação

```
gAgrupL.agg([min,max,sum,"mean"])
```

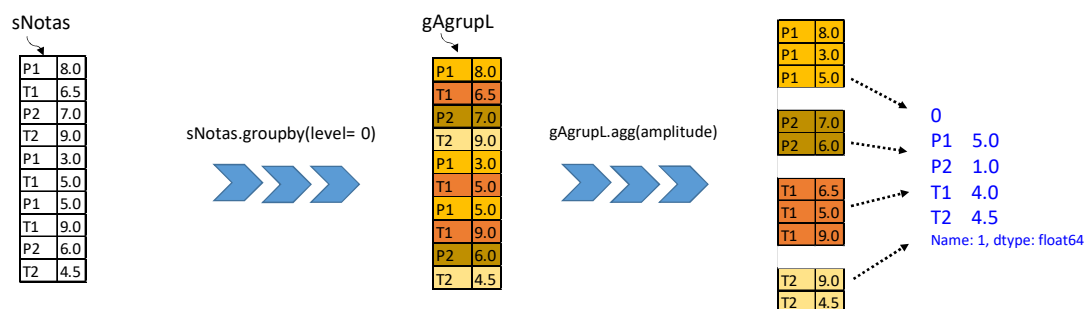


127

## Exemplo Funções de Agregação

```
def amplitude(x):
    return max(x) - min(x)

sNotas = pd.read_excel("notas.xlsx", header=None, index_col=0, squeeze=True)
gAgrupL=sNotas.groupby(level=0)
gAgrupL.agg(amplitude)
```



128





## Exercício Agrupamentos

Desenvolver as funções para responder as questões:

- I. Qual o total de gastos do Gasparzinho nos dias úteis (segunda à sexta) e o total no final de semana? Qual a relação percentual entre eles?
- II. Qual o gasto médio nos dias úteis e no final de semana?
- III. Quantos gastos ocorreram por dia?
- IV. Qual o maior valor gasto? Quantas ocorrências deste valor?
- V. Dividir os valores gastos em 3 grupos: Baixo (até R\$ 5,00), Médio (entre R\$ 5,01 e R\$ 20,00) e Alto (acima de R\$ 20,00) e mostrar:
  - quantidade de ocorrências em cada grupo
  - valor médio de cada grupo
  - relação percentual entre eles, mostrando-a também graficamente (gráfico de pizza)

129



## Agrupando pelo índice

Sab	76.80
Dom	71.22
Qua	65.18
Qui	65.23
Sex	44.58
Qui	41.52
Ter	12.74
Sex	3.64
Seg	35.28
Dom	69.32
Seg	92.32
Qui	34.91
Qui	93.86
Qua	37.35
Qui	85.13
Sab	56.16
Sab	64.50
Dom	93.51
Ter	83.15
Sex	85.40

groupby  
level= 0



Sab	76.80
Dom	71.22
Qua	65.18
Qui	65.23
Sex	44.58
Qui	41.52
Ter	12.74
Sex	3.64
Seg	35.28
Dom	69.32
Seg	92.32
Qui	34.91
Qui	93.86
Qua	37.35
Qui	85.13
Sab	56.16
Sab	64.50
Dom	93.51
Ter	83.15
Sex	85.40

método



Sab	76.80
Sab	56.16
Sab	64.50



Dom	71.22
Dom	69.32
Dom	93.51



Seg	35.28
Seg	92.32



Ter	12.74
Ter	83.15



Qua	65.18
Qua	37.35



Qui	65.23
Qui	41.52
Qui	34.91
Qui	93.86
Qui	85.13



Sex	44.58
Sex	3.64
Sex	85.40



130



## Agrupando por tipo de dia útil ou final de semana

Sab	76.80
Dom	71.22
Qua	65.18
Qui	65.23
Sex	44.58
Qui	41.52
Ter	12.74
Sex	3.64
Seg	35.28
Dom	69.32
Seg	92.32
Qui	34.91
Qui	93.86
Qua	37.35
Qui	85.13
Sab	56.16
Sab	64.50
Dom	93.51
Ter	83.15
Sex	85.40

groupby  
função



Sab	76.80
Dom	71.22
Qua	65.18
Qui	65.23
Sex	44.58
Qui	41.52
Ter	12.74
Sex	3.64
Seg	35.28
Dom	69.32
Seg	92.32
Qui	34.91
Qui	93.86
Qua	37.35
Qui	85.13
Sab	56.16
Sab	64.50
Dom	93.51
Ter	83.15
Sex	85.40

método



Sab	76.80
Sab	56.16
Sab	64.50
Dom	71.22
Dom	69.32
Dom	93.51



Seg	35.28
Seg	92.32
Ter	12.74
Ter	83.15
Qua	65.18
Qua	37.35
Qui	65.23
Qui	41.52
Qui	34.91
Qui	93.86
Qui	85.13
Sex	44.58
Sex	3.64
Sex	85.40



131



## Método 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) *localização* do arquivo: composto pelo caminho (absoluto/relativo) e nome

**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.

**s1:**

```
1 10
2 30
3 20
1 40
2 60
3 50
```

```
>>>pd.cut(s1,bins=3)
1      (9.95, 26.667]
2      (26.667, 43.333]
3      (9.95, 26.667]
1      (26.667, 43.333]
2      (43.333, 60.0]
3      (43.333, 60.0]
dtype: category
Categories (3,
      interval[float64]):
      [(9.95, 26.667] < (26.667,
      43.333] < (43.333, 60.0]]
```

```
>>>pd.cut(s1,bins=[10,20,70])
1      NaN
2      (20, 70]
3      (10, 20]
1      (20, 70]
2      (20, 70]
3      (20, 70]
dtype: category
Categories (2,
      interval[int64]): [(10,
      20] < (20, 70]]
```

```
>>>pd.cut(s1,bins=[10,20,70],
      include_lowest=True)
1      (9.999, 20.0]
2      (20.0, 70.0]
3      (9.999, 20.0]
1      (20.0, 70.0]
2      (20.0, 70.0]
3      (20.0, 70.0]
dtype: category
Categories (2,
      interval[float64]):
      [(9.999, 20.0] < (20.0,
      70.0]]
```