Data Wrangling: Conceitos e Aspectos Práticos

Eduardo Corrêa Gonçalves (ENCE/IBGE)
eduardo.correa@ibge.gov.br

Resumo. Ciência de dados é a disciplina preocupada com a análise e extração de conhecimento e informação a partir de bases de dados. A etapa de pré-processamento, onde as bases de dados relevantes devem ser reunidas e adequadamente formatadas, costuma ser a mais trabalhosa, ocupando tipicamente 80% do tempo consumido. É nesta fase que são realizadas as tarefas de seleção, limpeza e transformação dos dados, comumente referenciadas como atividades de *data wrangling*. Este minicurso possui como objetivo apresentar as principais informações necessárias para que o aluno possa conduzir processos práticos de data wrangling utilizando a biblioteca 'pandas' – um software livre, do tipo *open source*, que ao longo dos últimos anos se consolidou como a biblioteca para ciência de dados mais utilizada no ambiente Python.

1. Introdução

1.1 O que é Ciência de Dados?

Ciência de dados (*data science*) é a disciplina que combina ideias da Estatística e da Ciência da Computação para resolver o problema da descoberta de conhecimento em bases dados [Hand, 2020]. Nesta parceria, a Estatística tem o papel de fornecer as ferramentas para descrever, analisar, resumir, interpretar e realizar inferências sobre os dados. Por sua vez, a Ciência da Computação preocupa-se em oferecer tecnologias eficientes para o armazenamento, acesso, integração e transformação dos dados. Ou seja, o papel da Ciência da Computação é tornar viável a análise de bases de dados, muitas vezes complexas e volumosas, através de processos estatísticos [Corrêa, 2019]. Tipicamente, os projetos de ciência de dados são divididos nas quatro etapas de execução apresentadas na Figura 1 [Corrêa, 2020].

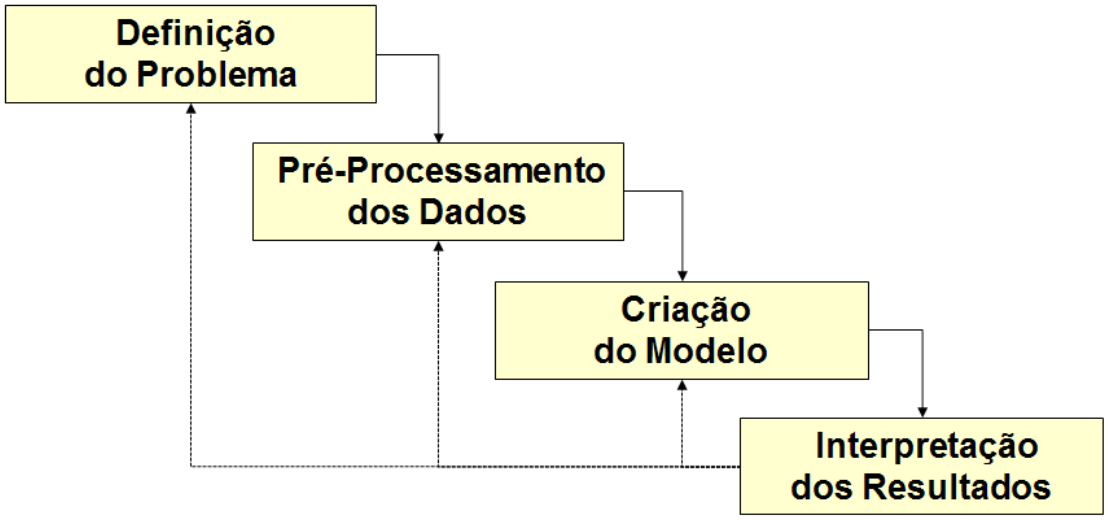


Figura 1. Etapas de um típico processo de ciência de dados

- Etapa 1: consiste simplesmente na definição do problema que será resolvido. Por exemplo: criar um sistema de recomendação para uma loja virtual (sistema que, de maneira autônoma, sugere produtos/serviços para os usuários).

- Etapa 2: execução das atividades de pré-processamento, onde as bases de dados relevantes (base de vendas, base de clientes etc.) devem ser reunidas, estudadas e adequadamente transformadas e formatadas. Esta etapa é também chamada de etapa de *data wrangling* ou *data munging* [Zhang et al., 2003].

- Etapa 3: execução de um ou mais algoritmos sobre os dados pré-processados, com o objetivo de extrair um modelo de ciência de dados. Considerando o exemplo do sistema de recomendação, seria possível empregar um algoritmo de análise de agrupamentos ou de mineração de regras de associação, entre outros, para criar o modelo [Han et al., 2011].

- Etapa 4: especialistas avaliam a qualidade modelo, procurando determinar a sua relevância e validade.

1.2 O que é Data Wrangling?

De acordo com a literatura, a etapa de pré-processamento ou data wrangling (etapa 2) costuma ser a mais trabalhosa em qualquer projeto relacionado à ciência de dados, ocupando tipicamente 80% do tempo consumido [Kandel et al., 2011; Press, 2016; Ruiz, 2017]. É durante esta etapa que são executadas as tarefas de **seleção**, **limpeza** e **transformação** dos dados que serão utilizados pelo algoritmo de construção do modelo.

1.2.1 Seleção de Dados

O objetivo da seleção de dados é coletar e reunir todos os dados que sejam relevantes para a resolução do problema de ciência de dados definido (por exemplo, combinar dados dos sistemas corporativos da empresa com dados disponibilizados na Internet). Por exemplo, se uma empresa deseja criar um sistema de recomendação sofisticado, poderia ser necessário selecionar as fontes de dados mostradas na Figura 2:

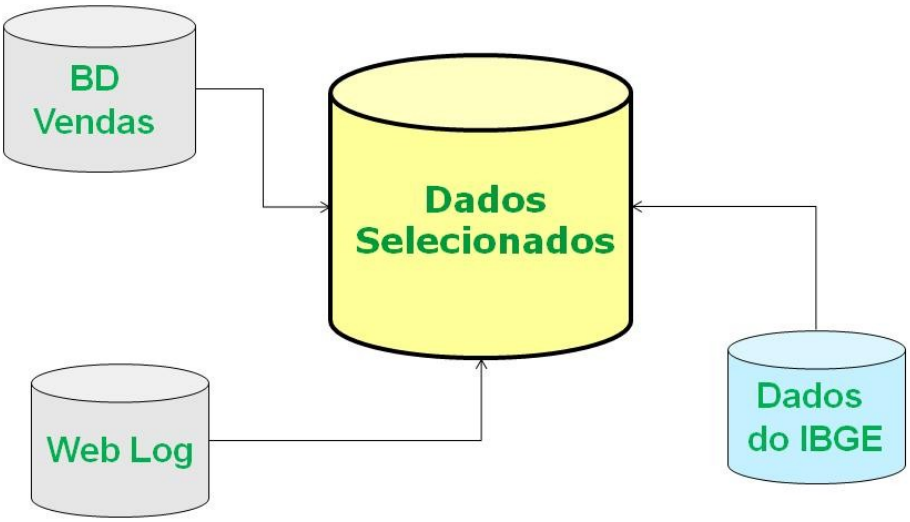


Figura 2. Seleção de Dados

- **Dados do sistema de vendas**: base que contém os dados demográficos do cliente (nome, sexo, endereço) e as informações de suas compras (produtos comprados, forma de pagamento, valor gasto).

- **Log de uso do servidor Web**: arquivo que armazena a sequência de páginas visitadas pelos clientes em cada sessão.

- **Dados externos**: muitas vezes pode ser interessante juntar também dados de fontes externas. Por exemplo, incorporar dados do IBGE para obter informações como a renda média e o número de habitantes das cidades dos clientes.

1.2.2 Limpeza e Transformação dos Dados

Limpeza, significa eliminar sujeira e informações irrelevantes. Transformação consiste em converter os dados de origem para um outro formato, mais adequado para ser usado pelo algoritmo que construirá o modelo. Alguns exemplos de situações práticas onde é necessário aplicar estes procedimentos são relacionados a seguir:

- **Dados contraditórios**. Ex.: idade igual a 6 anos e escolaridade Doutorado.

- **Valores com frequência acima da expectativa**. Ex.: muitas pessoas que se cadastraram em um sistema Web declararam morar em Araruama (Figura 3). Na verdade, por pressa ou distração, escolheram a primeira opção do formulário de cadastramento.

Figura 3. Valor com frequência acima da expectativa.

- **Transformar variável para formato mais adequado**. Ex.: transformar "Data de Nascimento" em "Idade".

- **Normalização de dados**. Ex.: mapear todos os valores numéricos para um número entre 0 e 1, para assegurar que os números grandes não dominem os números pequenos durante as operações matemáticas realizadas pelo algoritmo (por exemplo, para que a renda, que terá valores acima de R$1.000,00, não domine a idade com valores inferiores a 100).

- **Falta de uniformidade** entre as diferentes bases reunidas no processo de seleção. Ex.: alguns arquivos fonte onde atributo "Casado" é representado por 'S' (sim) e 'N' (não) e outros onde é representado por '1' e '0'.

- **Valores nulos**. Ex.: cliente não preencheu o valor de sua renda. Neste caso, será preciso adotar alguma técnica para imputar um valor, como utilizar o valor mais frequente ou o valor médio do estrato ao qual o cliente pertence.

1.3 Objetivos deste Minicurso

Data wrangling pode ser formalmente definido como um processo iterativo de exploração e transformação de dados com o intuito de habilitá-los para a análise [Kandel et al., 2011]. Este minicurso fornece ao aluno uma introdução aos conceitos e técnicas fundamentais de data wrangling através de exemplos práticos desenvolvidos com a biblioteca 'pandas' [Corrêa, 2020; McKinney, 2017; McKinney et al., 2020]. Trata-se de um software livre, do tipo *open source*, que ao longo dos últimos anos se consolidou como uma das bibliotecas para ciência de dados e data wrangling mais utilizadas em todo o mundo.

O texto do minicurso está dividido da seguinte forma. A Seção 2 aborda a biblioteca 'NumPy', voltada para a computação de alto desempenho sobre vetores e matrizes. O principal objetivo da seção é apresentar a técnica conhecida como computação vetorizada, que é muito utilizada em programas de ciência de dados. A Seção 3 introduz a biblioteca 'pandas', apresentando as técnicas elementares para criar e manipular o DataFrame, sua principal estrutura de dados. A Seção 4 é a principal do trabalho, onde os principais conceitos e técnicas de data wrangling são apresentados de forma prática, através de *scripts* desenvolvidos com o uso da 'pandas'. Encerrando o trabalho, a Seção 5 apresenta um exemplo simples de utilização da 'pandas' em um processo de machine learning.

É importante ressaltar que este minicurso é introdutório e possui enfoque prático. Para um melhor conhecimento teórico sobre data wrangling, o leitor pode recorrer às referências [Han et al. 2011; Kandel et al., 2011; Zhang et al., 2003; Witten et al., 2016].

2. NumPy

2.1 O que é NumPy?

A biblioteca 'NumPy' (*Numerical Python*) [NumPy, 2020] estende a linguagem Python com a estrutura de dados ndarray (*n-dimensional array*), direcionada para a computação de alto desempenho sobre vetores e matrizes. Apesar de, isoladamente, esta biblioteca fornecer poucas funções de alto nível para data wrangling, ela é considerada a "pedra fundamental" da computação científica em Python pelo fato de as suas propriedades e métodos terem sido utilizados como base para o desenvolvimento de diversas outras bibliotecas importantes para ciência de dados, como a própria 'pandas'. Esta seção apresenta uma breve introdução à 'NumPy', cobrindo os seguintes

tópicos: criação de vetores e matrizes, computação vetorizada e uso de funções matemáticas e estatísticas.

2.2. Criação de Vetores e Matrizes

A forma mais simples para criar um vetor 'NumPy' é chamando o método `np.array()`. No exemplo a seguir, apresenta-se o código para criar, extrair as propriedades e manipular os elementos do vetor apresentado na Figura 4. Algumas explicações são apresentadas nos comentários especificados no código.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
7.8	8.5	10.0	9.2	5.0	8.5	6.4	8.6	7.5	9.0

Figura 4. Vetor com 10 posições (notas de 10 alunos).

```
#P01: Olá Vetor NumPy!
import numpy as np #importa a biblioteca, renomeando-a para np

#Criamos um vetor a partir de uma lista
vet_notas = np.array([7.8, 8.5, 10.0, 9.2, 5.0, 8.5, 6.4, 8.6, 7.5, 9.0])

print('vet_notas = ', vet_notas) #imprime o vetor
print('type(vet_notas) = ', type(vet_notas)) #imprime o tipo (ndarray)
print('vet_notas.dtype = ', vet_notas.dtype) #prop. "dtype" - tipo dado (float64)
print('vet_notas.ndim = ', vet_notas.ndim)   #prop. "ndim" - num. dimensões (1)
print('vet_notas.shape = ', vet_notas.shape) #prop. "shape" - num posições (10)

#indexação básica
print('----------------------------------------')
print('primeiro elemento = ', vet_notas[0])               #indexação
print('último elemento = ', vet_notas[len(vet_notas)-1])  #indexação
print('3° e 4° elementos = ', vet_notas[2:4])             #fatiamento

#modifica a nota do quarto aluno
vet_notas[3] = 9.5

#imprime o novo vetor
print('----------------------------------------')
print('vet_notas novo = ', vet_notas)
```

```
>>> %Run p01_numpy_ola_vetor.py
vet_notas =  [ 7.8  8.5 10.   9.2  5.   8.5  6.4  8.6  7.5  9. ]
type(vet_notas) =  <class 'numpy.ndarray'>
vet_notas.dtype =  float64
vet_notas.ndim =  1
vet_notas.shape =  (10,)
----------------------------------------
primeiro elemento =  7.8
último elemento =  9.0
3o e 4o elementos =  [10.   9.2]
----------------------------------------
vet_notas novo =  [ 7.8  8.5 10.   9.5  5.   8.5  6.4  8.6  7.5  9. ]
```

Para a criação de matrizes, o método mais utilizado é o `np.reshape()`, que transforma um vetor em uma matriz de acordo com o número de linhas e colunas especificados como parâmetros. No exemplo a seguir, apresenta-se o código para criar, extrair as propriedades e manipular os elementos da matriz 2 x 3 apresentada abaixo.

```
| 7    8     9 |
|10   11    12 |
```

```
#P02: Olá Matriz NumPy!
import numpy as np

m = np.arange(7,13) #cria o vetor [7,8,9,10,11,12]
m = m.reshape((2,3))    #transforma-o em uma matriz 2x3

print('m = ', m)              #imprime a matriz
print('type(m) = ', type(m)) #imprime o tipo (ndarray)
print('m.dtype = ',m.dtype)  #prop. "dtype" - tipo do dado (int32 ou int64)
print('m.ndim = ',m.ndim)    #prop. "ndim" - número de dimensões (2)
print('m.shape = ',m.shape)  #propriedade "shape" - tot linhas x tot cols (2,3)

#indexação e fatiamento básicos
print('-----------------------------------------')
print('m[0,1] = ', m[0,1])       #[8] => (célula da 1ª linha, 2ª coluna)
print('m[1,:] = ', m[1,:])       #[10 11 12] => (toda 2ª linha)
print('m[:,2] = ', m[:,2])       #[9 12] => (toda 3ª coluna)
print('m[-1,-2:] = ', m[-1,-2:]) #[11 12] => (última linha, 2 últ. colunas)

#modifica o valor da célula [1,2] (2ª linha, 3ª coluna)
m[1,2] = 999
print('-----------------------------------------')
print('m nova = ', m) #imprime a nova matriz
```

```
>>> %Run p02_numpy_ola_matriz.py
m =  [[ 7  8  9]
 [10 11 12]]
type(m) =  <class 'numpy.ndarray'>
m.dtype =  int32
m.ndim =  2
m.shape =  (2, 3)
-----------------------------------------
m[0,1] =  8
m[1,:] =  [10 11 12]
m[:,2] =  [ 9 12]
m[-1,-2:] =  [11 12]
-----------------------------------------
m nova =  [[  7   8   9]
 [ 10  11 999]]
```

2.3. Computação Vetorizada

Em grande parte das situações práticas não é necessário utilizar os comandos `for` ou `while` para realizar cálculos sobre as estruturas da 'NumPy' e da 'pandas'. Isto porque, a maioria das funções pode ser executada através do mecanismo conhecido como **computação vetorizada** (*vectorization*). Neste processo, as operações são realizadas sobre cada elemento do vetor ou matriz automaticamente, sem a necessidade de programar um laço. Alguns exemplos:

- Se x é um vetor, e fazemos x * 2, obteremos como resultado um vetor que conterá todos os elementos de x multiplicados por 2.

- Ao efetuarmos uma soma de duas matrizes m1 e m2 com os shapes compatíveis, teremos como resultado uma nova matriz onde o valor da célula de índice [0,0] será igual a m1[0,0] + m2[0,0]; o valor da célula de índice [0,1] será m1[0,1] + m2[0,1], e assim sucessivamente (o mesmo vale para subtração, multiplicação ou divisão). Se os shapes não forem compatíveis, ocorrerá um erro do tipo "ValueError".

O programa a seguir exemplifica o conceito apresentado. Ele computa o valor da série S = (1/1) + (3/2) + (5/3) + (7/4) + … + (99/50) sem nenhum a implementação de um laço.

```
#P03: Cálculo da série S = (1/1) + (3/2) + (5/3) + (7/4) + … + (99/50)
#     sem programar laço (computação vetorizada)
import numpy as np

numerador   = np.arange(1,100,2)  #[1 3 5 ... 99]
denominador = np.arange(1,51)     #[1 2 3 ... 50]

S = sum(numerador / denominador)  # Σ[1/1 3/2 5/3 ... 99/50]

print('* * * resposta: s = {:.2f}'.format(S))

>>> %Run p03_numpy_calculo_serie.py
* * * resposta: s = 95.50
```

2.4. Funções Matemáticas e Estatísticas

Diversas funções matemáticas e estatísticas que operam de forma vetorizada podem ser aplicadas sobre ndarrays. As Tabelas 1 e 2 relacionam alguma das principais funções matemáticas e estatísticas, respectivamente (considere que "a" é um ndarray).

Tabela 1. Algumas funções matemáticas da NumPy

`np.abs(a)`: valor absoluto.
`np.ceil(a)`: arredondamento "para cima", ou seja, retorna o menor inteiro com valor igual ou superior ao valor da célula.
`np.floor(a)`: arredondamento "para baixo", ou seja, retorna o maior inteiro com valor igual ou inferior ao valor da célula.
`np.rint(a)`: arredondamento para o inteiro mais próximo, preservando o "dtype".
`np.sqrt(a)`: raiz quadrada.
`np.square(a)`: eleva ao quadrado.
`np.exp(a)`: computa e^x para cada elemento x do array "a".
`np.log(a)`, `np.log10(a)`, `np.log10(a)`: logaritmo natural (base e), na base 10 e na base 2.
`np.isnan(a)`: retorna um array booleano que indica, para cada célula de "a", se a mesma armazena um valor `NaN` (`True`) ou não (`False`);
`np.isfinite(a)`: retorna um array booleano que indica, para cada célula de "a", se a mesma armazena

um valor finito (diferente de `NaN`);
`np.`**`isfinite`**`(a), np.`**`isinf`**`(a)`: retorna um array booleano que indica, para cada célula de "a", se a mesma armazena um valor infinito;
`np.`**`cos`**`(a), np.`**`sin`**`(a), np.`**`tan`**`(a), np.`**`arccos`**`(a), np.`**`arcsin`**`(a), np.`**`arctan`**`(a)`: funções trigonométricas;

Tabela 2. Algumas funções estatísticas da NumPy

`np.`**`sum`**`()`: computa a soma de todos os elementos em um array ou de um de seus eixos;
`np.`**`mean`**`()`: computa média de todos os elementos em um array ou de seus eixos (células com valor `NaN` são ignoradas);
`np.`**`var`**`()` e `np.`**`std`**`()`: variância e desvio padrão, com graus de liberdade podendo ser ajustados via parâmetro (o valor *default* é n);
`np.`**`min`**`()` e `np.`**`max`**`()`: menor e maior valor de um array ou de um de seus eixos;
`np.`**`argmin`**`()` e `np.`**`argmax`**`()`: índice do menor e maior valor;

Trabalhar com funções matemáticas e estatísticas sobre vetores é trivial. No entanto, para trabalhar com as mesmas funções sobre matrizes, é preciso conhecer o conceito de "axis" (eixo). Basicamente, trata-se de um parâmetro que indica se as agregações devem ser obtidas por coluna (nesse caso, especifica-se axis=0) ou por linha (nesse caso, especifica-se axis=1). O programa a seguir mostra como aplicar funções estatísticas sobre a matriz "notas", que armazena as notas de quatro alunos em três diferentes provas (cada linha corresponde a um aluno e cada coluna a uma prova).

```
|9.8  7.2   8.0|
|5.3  4.0   3.5|
|5.5  8.1   7.2|
|7.8  7.5   6.5|
```

```
#P04: Estatísticas sobre matrizes + Axis
import numpy as np

#cria a matriz "notas"
notas = np.array([9.8, 7.2, 8.0, 5.3, 4.0, 3.5, 5.5, 8.1, 7.2, 7.8, 7.5, 6.5])
notas = notas.reshape((4,3))

#imprime a matriz e gera as estatísticas
print('notas: ', notas)

print('média geral: ', notas.mean())
print('média de cada prova: ', notas.mean(axis=0))
print('média de cada aluno: ', notas.mean(axis=1))

print('maior nota geral: ', notas.max())
print('maior nota de cada prova: ', notas.max(axis=0))
print('maior nota de cada aluno: ', notas.max(axis=1))
```

```
>>> %Run p04_numpy_estat_matriz.py
notas:  [[9.8 7.2 8. ]
 [5.3 4.  3.5]
 [5.5 8.1 7.2]
 [7.8 7.5 6.5]]
média geral:  6.7
média de cada prova:  [7.1 6.7 6.3]
média de cada aluno:  [8.33333333 4.26666667 6.93333333 7.26666667]
maior nota geral:  9.8
maior nota de cada prova:  [9.8 8.1 8. ]
maior nota de cada aluno:  [9.8 5.3 8.1 7.8]
```

Esta seção apresentou uma breve introdução à biblioteca 'NumPy', focando apenas nas operações matemáticas e estatísticas com o uso da computação vetorizada. Para conhecer outras funcionalidades da biblioteca, consulte [Corrêa, 2019; McKinney, 2017; NumPy 2020].

3. Introdução à Pandas

A biblioteca 'pandas' (*Python Data Analysis Library*) [Corrêa, 2020; McKinney, 2017; McKinney et al., 2020] é a mais importante e popular biblioteca para ciência de dados do Python. As funcionalidades por ela oferecidas consistem basicamente em uma combinação de técnicas eficientes para processamento de vetores e matrizes, com um conjunto de funções específicas para a manipulação e transformação de dados tabulares.

A 'pandas' oferece duas estruturas de dados: **Series** (para estruturas dados de séries temporais) e **DataFrame** (para estruturar dados tabulares). Esta introduz ambas as estruturas.

3.1 Series: Criação e Operações Básicas

Uma Series pode ser entendida como **vetor 'Numpy' de dados** associado a um **vetor de rótulos** (também denominado vetor de índices, ou simplesmente índices). Ou seja: Series = vetor de dados + vetor de rótulos. No exemplo a seguir, apresenta-se o código para criar, extrair as propriedades e manipular os elementos do vetor apresentado na Figura 4. Algumas explicações são apresentadas nos comentários especificados no código.

[BR]	[FR]	[UK]	[IT]	[US]
Real	Euro	Libra	Euro	Dólar

Figura 5. Series onde os índices são siglas e os valores o nome da moeda de países

```
#P05: Series de países e moedas
import pandas as pd

#(1)-criação da Series onde índices=siglas de países e valores=nome da moeda
siglas = ['BR','FR','UK','IT','US']
moedas = ['Real','Euro','Libra','Euro','Dólar']
s2 = pd.Series(moedas,index=siglas)

print('-----------------------------------------')
print('s2:')
print(s2)

#(2)-podemos utilizar os rótulos para indexar
print('-----------------------------------------')
```

```
print(s2['UK'])           #'Libra'
print('\n')
print(s2[['BR','IT']])    #['BR':'Real', 'IT':'Euro']
print('\n')
print(s2[1:3])            #['FR':'Euro', 'UK':'Libra']
print('\n')
print('BR' in s2)         #True
print('AR' in s2)         #False

#(3)-As propriedades values e index retornam, respectivamente, os valores
#    e índices da Series, respectivamente
print('-----------------------------------------')
print(s2.values)          #['Real' 'Euro' 'Libra' 'Euro' 'Dólar']
print(s2.index)           #Index(['BR', 'FR', 'UK', 'IT', 'US'], dtype='object')
```

```
>>> %Run p05_series_paises_moedas.py
-----------------------------------------
s2:
BR     Real
FR     Euro
UK    Libra
IT     Euro
US    Dólar
dtype: object
-----------------------------------------
Libra


BR    Real
IT    Euro
dtype: object


FR     Euro
UK    Libra
dtype: object


True
False
-----------------------------------------
['Real' 'Euro' 'Libra' 'Euro' 'Dólar']
Index(['BR', 'FR', 'UK', 'IT', 'US'], dtype='object')
```

3.3 DataFrame: Criação e Operações Básicas

O DataFrame é a estrutura de dados utilizada para representar dados tabulares em memória, isto é, dados dispostos em **linhas** e **colunas** (ambas podendo ser indexadas). Cada coluna de um DataFrame corresponde a uma Series (por isso, o DataFrame pode ser visto como um dicionário de Series). A seguir, apresenta-se o código de um programa que cria o DataFrame "df_pessoas" (Figura 6), que possui sete linhas e quatro colunas. O programa mostra ainda a forma de atribuir rótulos para linhas e colunas e também os métodos para indexar elementos do DataFrame (`iloc` e `loc`).

	nome	idade	salario	uf
P1	Alex	50	5000	RJ
P2	Carlos	21	2000	RJ
P3	Jane	55	3500	SP
P4	Rakesh	37	6500	SP
P5	Elis	18	2000	RJ
P6	Isabel	42	4500	MG
P7	Andres	33	3500	SP

Figura 6. DataFrame com dados de sete diferentes pessoas

```
#P06: Olá DataFrame
import pandas as pd

#(1)-Há várias maneiras de construir DataFrames.
#    No exemplo abaixo, o fazemos a partir de um dicionário de listas
dados = {'nome': ['Alex','Carlos','Jane','Rakesh','Elis','Isabel','Andres'],
         'idade': [50,21,55,37,18,42,33],
         'sal': [5000,2000,3500,6500,2000,4500,3500],
         'uf': ['RJ','RJ','SP','SP','RJ','MG','SP']}

df_pessoas = pd.DataFrame(dados)

print('(1)-DataFrame original\n')
print(df_pessoas)           #imprime todo o DataFrame

#(2)-Atribui nomes para os índices (método index)
#    e modifica os nomes das colunas (método columns)
#    (agora sim vai ficar com o formato igual ao da Figura 22)
print('-----------------------------------------')
print('(2)-DataFrame com novos nomes de índices e colunas\n')
df_pessoas.index = ['P1','P2','P3','P4','P5','P6','P7']
df_pessoas.columns = ['nome','idade','salario','uf']
print(df_pessoas)

#(3)-INDEXAÇÃO
#[3.1]-Uma LINHA inteira pode ser recuperada pelo seu rótulo ou posição,
#      através dos métodos "loc" e "iloc", respectivamente
print('-----------------------------------------')
print('(3)-Indexação de linhas, colunas e células\n')
print(df_pessoas.loc['P5'])   #recupera a linha com rótulo 'P5' (Elis)
print('\n')
print(df_pessoas.iloc[0]) #recupera a linha na posição 0 (primeira linha - Alex)

#[3.2]-Uma COLUNA inteira pode ser recuperada como uma Series
#      utilizando a notação estilo "dicionário" ou estilo "atributo".
#      A Series retornada terá o mesmo índice do DataFrame
print('-----------------------------------------')
print(df_pessoas['idade'])   #recupera a coluna "idade" (notação "dicionário")
print('\n')
print(df_pessoas.nome)       #recupera a coluna "nome" (notação "atributo")

#[3.3]-Uma CÉLULA pode ser recuperada de diferentes formas.
```

```
#      Abaixo, utilizamos os métodos "iloc" e "loc" novamente
print('-----------------------------------------')
print(df_pessoas.iloc[4][2])           #posição da linha + posição da coluna
print(df_pessoas.iloc[4]['salario'])   #posição da linha + rótulo da coluna
print(df_pessoas.loc['P5'][2])         #rótulo da linha  + posição da coluna
print(df_pessoas.loc['P5']['salario']) #rótulo da linha  + rótulo da coluna
```

```
>>> %Run p06_dataframe_pessoas.py
(1)-DataFrame original

     nome  idade   sal  uf
0    Alex     50  5000  RJ
1  Carlos     21  2000  RJ
2    Jane     55  3500  SP
3  Rakesh     37  6500  SP
4    Elis     18  2000  RJ
5  Isabel     42  4500  MG
6  Andres     33  3500  SP
-----------------------------------------
(2)-DataFrame com novos nomes de índices e colunas

      nome  idade  salario  uf
P1    Alex     50     5000  RJ
P2  Carlos     21     2000  RJ
P3    Jane     55     3500  SP
P4  Rakesh     37     6500  SP
P5    Elis     18     2000  RJ
P6  Isabel     42     4500  MG
P7  Andres     33     3500  SP
-----------------------------------------
(3)-Indexação de linhas, colunas e células

nome       Elis
idade        18
salario    2000
uf           RJ
Name: P5, dtype: object


nome       Alex
idade        50
salario    5000
uf           RJ
Name: P1, dtype: object
-----------------------------------------
P1    50
P2    21
P3    55
P4    37
P5    18
P6    42
P7    33
Name: idade, dtype: int64


P1      Alex
P2    Carlos
P3      Jane
P4    Rakesh
```

```
P5      Elis
P6    Isabel
P7    Andres
Name: nome, dtype: object
----------------------------------------
2000
2000
2000
2000
>>>
```

Na prática, os dados dos DataFrames são obtidos a partir de bancos de dados ou arquivos. Por este motivo, a 'pandas' permite a importação de dados armazenados em virtualmente qualquer formato de arquivo (CSV, JSON, XML, etc.) ou SGBD (SQLite, MongoDB, etc.). O exemplo a seguir mostra como importar uma pequena base de dados estruturada em um arquivo CSV chamado "lojas.csv". Esta base armazena informações sobre treze lojas pertencentes a seis diferentes empresas:

```
raiz,sufixo,fantasia,po,salario,uf
22222222,0001,PYTHONSHOP I  ,20,35000,SP
22222222,0002,PYTHONSHOP II ,5,4000,RJ
22222222,0003,PYTHONSHOP III,15,31000,MG
33333333,0001,NPSERV LOJA A ,8,9200,MG
33333333,0002,NPSERV LOJA B ,0,0,MG
44444444,0001,XYZ MATRIZ    ,112,250000,RJ
44444444,0003,XYZ CENTRO    ,101,103900,SP
44444444,0004,XYZ NORTE     ,48,60000,RJ
44444444,0005,XYZ SUL       ,50,65000,RJ
44444444,0010,XYZ LITORAL   ,1,2500,RJ
55555555,0001,JAVASERV      ,40,40000,SP
66666666,0001,CAFEDB        ,12,12900,RS
99999999,0001,PANDAS KING   ,1,3000,MG
```

A base de dados possui seis colunas:

- **raiz**: raiz do CNPJ, código de 8 dígitos que identifica unicamente uma empresa.
- **sufixo**: sufixo do CNPJ. Junto com a raiz, identifica unicamente uma unidade local.
- **fantasia**: nome fantasia da loja (veja que possui espaços em branco à direita).
- **po**: pessoal ocupado, ou seja, a quantidade de funcionários que trabalham na loja.
- **salario**: valor mensal gasto pela loja com o salário dos funcionários.
- **uf**: unidade da federação onde a loja está localizada.

A base pode ser importada com o uso do método **`read_csv(`**, como mostra o programa a seguir (Para conhecer outros parâmetros do método read_csv() e saber como importar outros tipos de arquivos, consulte [Corrêa, 2020]):

```
#P07: Importação de CSV padrão para DataFrame
import pandas as pd

#(1)-Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')
print(df_lojas)
```

```
#(2)-mostra o total de linhas e colunas
print('-------------------------------------')
num_linhas = df_lojas.shape[0]
num_colunas = df_lojas.shape[1]
print("número de linhas: ", num_linhas)
print("número de colunas: ", num_colunas)

#(3)-primeiras linhas - head()
print('-------------------------------------')
print("primeiras linhas\n: ", df_lojas.head())

#(4)-últimas linhas - tail()
print('-------------------------------------')
print("primeiras linhas\n: ", df_lojas.tail())
```

```
>        raiz  sufixo        fantasia   po  salario  uf
0   22222222       1  PYTHONSHOP I     20    35000  SP
1   22222222       2  PYTHONSHOP II     5     4000  RJ
2   22222222       3  PYTHONSHOP III   15    31000  MG
3   33333333       1  NPSERV LOJA A     8     9200  MG
4   33333333       2  NPSERV LOJA B     0        0  MG
5   44444444       1  XYZ MATRIZ      112   250000  RJ
6   44444444       3  XYZ CENTRO      101   103900  SP
7   44444444       4  XYZ NORTE        48    60000  RJ
8   44444444       5  XYZ SUL          50    65000  RJ
9   44444444      10  XYZ LITORAL       1     2500  RJ
10  55555555       1  JAVASERV         40    40000  SP
11  66666666       1  CAFEDB           12    12900  RS
12  99999999       1  PANDAS KING       1     3000  MG
-------------------------------------
número de linhas:  13
número de colunas:  6
-------------------------------------
primeiras linhas
:         raiz  sufixo        fantasia  po  salario  uf
0  22222222       1  PYTHONSHOP I    20    35000  SP
1  22222222       2  PYTHONSHOP II    5     4000  RJ
2  22222222       3  PYTHONSHOP III  15    31000  MG
3  33333333       1  NPSERV LOJA A    8     9200  MG
4  33333333       2  NPSERV LOJA B    0        0  MG
-------------------------------------
primeiras linhas
:          raiz  sufixo        fantasia  po  salario  uf
8   44444444       5  XYZ SUL         50    65000  RJ
9   44444444      10  XYZ LITORAL      1     2500  RJ
10  55555555       1  JAVASERV        40    40000  SP
11  66666666       1  CAFEDB          12    12900  RS
12  99999999       1  PANDAS KING      1     3000  MG
```

Na próxima seção, serão mostradas diferentes maneiras de executar operações de data wrangling sobre o DataFrame com os dados das lojas. Isto é, serão apresentadas algumas das ferramentas oferecidas pela pandas para explorar, limpar e transformar este DataFrame.

4. Data Wrangling com a 'pandas'

Os DataFrames 'pandas' possuem um rico conjunto de métodos para pré-processamento, filtragem, integração e exploração de bases de dados. Nesta seção, são apresentados diferentes exemplos.

4.1 Projeção

A projeção é a operação que gera um novo DataFrame a partir da extração de algumas colunas de um outro DataFrame. No contexto do data wrangling, a operação é útil para descartar atributos que não estarão envolvidos na criação do modelo (seja para redução da dimensionalidade do problema ou simplesmente para eliminar atributos irrelevantes, como um "id"). Para implementá-la basta passar uma lista com os atributos que deseja-se projetar:

```
#P08: Projeção
import pandas as pd

#Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

#gera um novo DataFrame apenas com a uf, po e salário (nesta ordem)
df_uf_po_sal = df_lojas[['uf','po','salario']]
print(df_uf_po_sal)
```

```
>>> %Run p08_projecao.py
    uf     po   salario
0   SP   20.0   35000.0
1   RJ    5.0    4000.0
2   MG   15.0   31000.0
3   MG    8.0    9200.0
4   MG    0.0       0.0
5   RJ  112.0  250000.0
6   SP  101.0  103900.0
7   RJ   48.0   60000.0
8   RJ   50.0   65000.0
9   RJ    NaN       NaN
10  SP   40.0   40000.0
11  RS   12.0   12900.0
12  MG    1.0    3000.0
```

4.2 Seleção

É a operação que gera um novo DataFrame a partir da extração de algumas linhas de interesse de um outro DataFrame. Também é conhecida como filtragem de linhas.

A técnica de filtragem mais utilizada é a **indexação booleana**, onde passamos para o DataFrame uma Series de elementos booleanos (True / False) com o objetivo de obter apenas as linhas associadas ao valor True. A Series booleana é, por sua vez, construída através de um teste lógico. No programa a seguir, utilizamos a operação de seleção para gerar dois diferentes DataFrames:

- "df_rj": apenas as lojas do RJ
- "df_rj_grandes": apenas as lojas do RJ com 50 ou mais funcionários.

```
#P09: Seleção
import pandas as pd
```

```
#Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

#gera um novo DataFrame contendo apenas as lojas do RJ
v = (df_lojas['uf']=='RJ') #gera a Series booleana
df_rj = df_lojas[v]        #filtra as linhas

#gera um novo DataFrame contendo apenas as lojas do RJ c/ 50 ou mais funcionários
v = (df_lojas['uf']=='RJ') & (df_lojas['po']>=50) #gera a Series booleana
df_rj_grandes = df_lojas[v]                       #filtra as linhas

#imprime os DataFrames
print(df_rj)
print('----------------------------------------')
print(df_rj_grandes)
```

```
>>> %Run p09_selecao.py
       raiz  sufixo        fantasia   po  salario  uf
1  22222222       2  PYTHONSHOP II     5     4000  RJ
5  44444444       1  XYZ MATRIZ      112   250000  RJ
7  44444444       4  XYZ NORTE        48    60000  RJ
8  44444444       5  XYZ SUL          50    65000  RJ
9  44444444      10  XYZ LITORAL       1     2500  RJ
----------------------------------------
       raiz  sufixo        fantasia   po  salario  uf
5  44444444       1  XYZ MATRIZ      112   250000  RJ
8  44444444       5  XYZ SUL          50    65000  RJ
```

4.3 Explorando uma Base de Dados

Uma das primeiras atividades realizadas em qualquer processo de data wrangling é o estudo das propriedades de cada atributo da base de dados através do emprego da estatística descritiva, tabulações e gráficos.

Para começar, o programa a seguir mostra como computar as estatísticas básicas (média, mediana, desvio padrão, etc.) de duas diferentes colunas do DataFrame "df_lojas": "salario" (numérica) e a coluna "uf" (categórica).

```
#P10: Estatísticas Básicas
import pandas as pd

#(1)-Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

#(2)-Estatísticas básicas da variável "salario" (numérica)
print('- Estatísticas da variável salário: ')
print('média        : {:.2f}'.format(df_lojas['salario'].mean()))
print('mediana      : {:.2f}'.format(df_lojas['salario'].median()))
print('variância    : {:.2f}'.format(df_lojas['salario'].var()))
print('desvio padrão: {:.2f}'.format(df_lojas['salario'].std()))

#(3)-Estatísticas básicas da variável "uf" (categórica)
print('\n- Estatísticas da variável uf: ')
print('moda: ', df_lojas['uf'].mode())
print('domínio: ', df_lojas['uf'].unique())#retorna todas as categorias distintas
print('freq. das categorias:')
print(df_lojas['uf'].value_counts()) #retorna a freq. de cada categoria
```

```
>>> %Run p10_estatisticas.py
- Estatísticas da variável salário:
média         : 47423.08
mediana       : 31000.00
variância     : 4662681923.08
desvio padrão: 68283.83

- Estatísticas da variável uf:
moda:  0    RJ
dtype: object
domínio:  ['SP' 'RJ' 'MG' 'RS']
freq. das categorias:
RJ    5
MG    4
SP    3
RS    1
Name: uf, dtype: int64
```

Agregação é uma operação que produz valores escalares a partir do processamento de grupos de linhas de um DataFrame [Corrêa, 2020]. Os grupos de linhas são definidos a partir de valores de um ou mais atributos categóricos, enquanto os valores escalares são computados sobre atributos numéricos.

As agregações na 'pandas' podem ser implementadas com o método **`groupby()`** que funciona de maneira similar ao GROUP BY da linguagem SQL. No programa a seguir, os grupos são definidos com o uso do atributo "uf" e os valores escalares computados sobre o atributo "po".

```
#P11: groupby()
import pandas as pd

#(1)-Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

#(2)-Gera uma variável "grouped" onde a chave é "uf" e a medida "po"
grupo_po_uf = df_lojas['po'].groupby(df_lojas['uf'])

#(3)-Computa agregados a partir da variável gerada
print('- Soma do PO, por UF: ',grupo_po_uf.sum())
print('-----------------------------------')
print('- Total de lojas, por UF:', grupo_po_uf.count())
```

```
>>> %Run p11_group_by.py
- Soma do PO, por UF:  uf
MG     24
RJ    216
RS     12
SP    161
Name: po, dtype: int64
-----------------------------------
- Total de lojas, por UF: uf
MG    4
RJ    5
RS    1
SP    3
Name: po, dtype: int64
```

O segredo do programa está na seguinte linha de código:

```
grupo_po_uf = df_lojas['po'].groupby(df_lojas['uf'])
```

Essa linha é responsável por criar uma variável ***grouped*** ou **objeto GroupBy** chamado "grupo_po_uf". Esta variável não armazena o resultado de nenhum cálculo, mas apenas informações que facilitarão a produção de resultados agregados da coluna "po" (variável numérica) por "uf" (variável categórica, denominada pela 'pandas' de **variável chave** do grupo). A partir de "grupo_po_uf" será possível produzir agregações de "po" por "uf" utilizando qualquer função estatística da NumPy. No exemplo apresentado, utilizamos as funções **`sum()`** e **`count()`** para obter, respectivamente, uma tabela com a soma de pessoal ocupado por UF e outra com o total de lojas por UF (veja que esse segundo resultado independe do "po").

Se você quiser agregar por mais de uma coluna, basta especificar uma lista como parâmetro do método **`groupby()`**. Por exemplo, para criar uma variável *grouped* cruzando po por raiz do cnpj e UF, você deve utilizar a sintaxe abaixo:

```
grupo_po_uf_raiz = df_lojas['po'].groupby([df_lojas['uf'], df_lojas['raiz']])
```

É possível também visualizar os dados de forma gráfica. A 'pandas' pode gerar boxplots, gráficos de barras, histogramas, gráficos de dispersão, gráficos de linha e outros. O código a seguir mostra como gerar um boxplot para a variável "salario". Através de um boxplot é possível visualizar em um mesmo gráfico a média, mediana, desvio padrão e *outliers* (valores suspeitos que podem ser normais ou representar algum erro na base de dados). A Figura 7 exibe o boxplot gerado pelo programa.

```
#P12: boxplot
import pandas as pd

#(1)-Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

#(2)-Gera o boxplot
boxplot = df_lojas.boxplot(column=['salario'], showmeans=True)
```

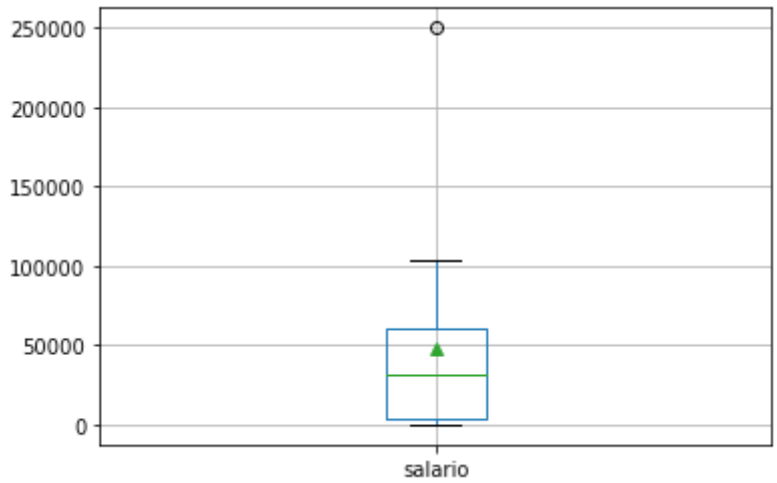


Figura 7. Boxplot do atributo "salario".

4.4 Discretização

Discretizar um atributo contínuo significa converter os seus valores para um conjunto reduzido de valores discretos ou categóricos. O próximo exemplo mostra como converter o atributo "po" para "faixa_po" a partir da definição de uma função e de sua aplicação com o emprego do método `apply()`.

```
#P12: Discretização
import pandas as pd

#(1)-Define uma função para discretizar o PO
def discretiza_po(po):
    if po == 0: return "0";
    elif po <= 10: return "1-10";
    elif po <= 50: return "11-50";
    elif po > 50: return ">50"
    else: return "desconhecido";

#(2)-Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

#(3)-Cria a coluna "fx_po"
df_lojas['faixa_po'] = df_lojas['po'].apply(discretiza_po)

#(4)-Projeta o nome fantasia, o PO e a faixa de PO
print(df_lojas[['fantasia','po','faixa_po']])

#(5)-Após a discretização, podemos remover o 'po' se quisermos
df_lojas = df_lojas.drop(columns=['po'])
print('-----------------------------------')
print('DataFrame após a exclusão do "po"'); print(df_lojas)
>>> %Run p13_discretizacao.py
          fantasia   po faixa_po
0   PYTHONSHOP I     20    11-50
1   PYTHONSHOP II     5     1-10
2   PYTHONSHOP III   15    11-50
3   NPSERV LOJA A     8     1-10
4   NPSERV LOJA B     0        0
5   XYZ MATRIZ      112      >50
6   XYZ CENTRO      101      >50
7   XYZ NORTE        48    11-50
8   XYZ SUL          50    11-50
9   XYZ LITORAL       1     1-10
10  JAVASERV         40    11-50
11  CAFEDB           12    11-50
12  PANDAS KING       1     1-10
-----------------------------------
DataFrame após a exclusão do "po"
        raiz  sufixo        fantasia  salario  uf faixa_po
0   22222222       1  PYTHONSHOP I      35000  SP    11-50
1   22222222       2  PYTHONSHOP II      4000  RJ     1-10
2   22222222       3  PYTHONSHOP III    31000  MG    11-50
3   33333333       1  NPSERV LOJA A      9200  MG     1-10
4   33333333       2  NPSERV LOJA B         0  MG        0
5   44444444       1  XYZ MATRIZ       250000  RJ      >50
6   44444444       3  XYZ CENTRO       103900  SP      >50
```

```
7   44444444       4  XYZ NORTE          60000  RJ    11-50
8   44444444       5  XYZ SUL            65000  RJ    11-50
9   44444444      10  XYZ LITORAL         2500  RJ     1-10
10  55555555       1  JAVASERV           40000  SP    11-50
11  66666666       1  CAFEDB             12900  RS    11-50
12  99999999       1  PANDAS KING         3000  MG     1-10
```

4.5 Normalização

Normalizar os atributos numéricos de uma base de dados significa colocá-los dentro de uma faixa comum, como [0.0, 1.0]. Este tipo de operação é feita para assegurar que os atributos que armazenam números grandes não dominem os que armazenam números pequenos durante a criação de um modelo. Trata-se de uma técnica de data wrangling especialmente importante para situações onde deseja-se utilizar algoritmos baseados em medidas de distância, como o kMeans e o k-NN [Han et al., 2011; Witten et al., 2016]. O exemplo a seguir mostra um programa que normaliza os atributos "po" e "salário".

```
#P14: Normalização
import pandas as pd

#(1)-Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

#(2)-Normaliza o "salario"
sal_max = max(df_lojas['salario'])
sal_min = min(df_lojas['salario'])
df_lojas['sal_norm'] = (df_lojas['salario'] - sal_min) / (sal_max - sal_min)

#(3)-Normaliza o "po"
po_max = max(df_lojas['po'])
po_min = min(df_lojas['po'])
df_lojas['po_norm'] = (df_lojas['po'] - po_min) / (po_max - po_min)

#(4)-Após a normalização, remove os atributos originais
df_lojas = df_lojas.drop(columns=['po','salario'])

#(5)-Imprime o DataFrame transformado
print('-----------------------------------')
print('DataFrame após a normalização das variáveis numéricas')
print(df_lojas)
```

```
>>> %Run p14_normalizacao.py
-----------------------------------
DataFrame após a normalização das variáveis numéricas
        raiz  sufixo        fantasia  uf  sal_norm   po_norm
0   22222222       1  PYTHONSHOP I    SP    0.1400  0.178571
1   22222222       2  PYTHONSHOP II   RJ    0.0160  0.044643
2   22222222       3  PYTHONSHOP III  MG    0.1240  0.133929
3   33333333       1  NPSERV LOJA A   MG    0.0368  0.071429
4   33333333       2  NPSERV LOJA B   MG    0.0000  0.000000
5   44444444       1  XYZ MATRIZ      RJ    1.0000  1.000000
6   44444444       3  XYZ CENTRO      SP    0.4156  0.901786
7   44444444       4  XYZ NORTE       RJ    0.2400  0.428571
8   44444444       5  XYZ SUL         RJ    0.2600  0.446429
9   44444444      10  XYZ LITORAL     RJ    0.0100  0.008929
10  55555555       1  JAVASERV        SP    0.1600  0.357143
```

```
11  66666666       1  CAFEDB          RS    0.0516  0.107143
12  99999999       1  PANDAS KING     MG    0.0120  0.008929
```

4.6 Conversão de Atributos Categóricos

Outro tipo de transformação muito utilizada consiste na conversão de um atributo categórico com *k* categorias para *k* atributos binários. Esse processo é exigido por alguns algoritmos de ciência de dados que suportam apenas atributos numéricos. O exemplo a seguir mostra o processo aplicado ao atributo "uf". No programa, este atributo categórico é convertido para quatro variáveis binárias: "uf_RJ", "uf_SP", "uf_MG" e "uf_RS".

```
#P15: get_dummies()
import pandas as pd

#(1)-Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

#(2)-gera k atributos binários a partir do atributo "uf"
dummies = pd.get_dummies(df_lojas['uf'], prefix="uf")
df_lojas = df_lojas.join(dummies)

#(3)-Projeta o nome fantasia, a uf e os novos atributos
print(df_lojas[['fantasia','uf','uf_RJ','uf_SP','uf_MG','uf_RS']])
```

```
>>> %Run p15_get_dummies.py
          fantasia  uf  uf_RJ  uf_SP  uf_MG  uf_RS
0    PYTHONSHOP I    SP      0      1      0      0
1    PYTHONSHOP II   RJ      1      0      0      0
2    PYTHONSHOP III  MG      0      0      1      0
3    NPSERV LOJA A   MG      0      0      1      0
4    NPSERV LOJA B   MG      0      0      1      0
5    XYZ MATRIZ      RJ      1      0      0      0
6    XYZ CENTRO      SP      0      1      0      0
7    XYZ NORTE       RJ      1      0      0      0
8    XYZ SUL         RJ      1      0      0      0
9    XYZ LITORAL     RJ      1      0      0      0
10   JAVASERV        SP      0      1      0      0
11   CAFEDB          RS      0      0      0      1
12   PANDAS KING     MG      0      0      1      0
```

4.7 Combinando DataFrames

O método `merge()` é um dos mais utilizados para realizar a tarefa de integração de dados em processos de data wrangling. Ele possibilita com que dois DataFrames possam ser combinados de forma semelhante à operação de junção da SQL. É possível utilizar as abordagens "inner", "outer" e "full". O exemplo a seguir, mostra como combinar as bases de dados "lojas.csv" e "emps.csv". O conteúdo da última é apresentado a seguir.

```
raiz,razao,natjur
11111111,UNIVERSIDADE ALPHA,1
22222222,PYTHONISTA,2
33333333,NPSERV,1
44444444,MERCADO XYZ,2
55555555,JAVANESE BRASIL,2
66666666,CAFEDB,2
88888888,ABG,1
```

A base de dados “emps.csv” armazena informações empresas que podem ser “donas” das lojas constantes na base de dados “lojas.csv”. Ela possui sete linhas (sete empresas) e três colunas. As colunas são descritas a seguir:

- **raiz**: raiz do CNPJ, código de 8 dígitos que identifica unicamente uma empresa.
- **razao**: razão social da empresa.
- **natjur:** código da natureza jurídica da empresa (1=Empresa Pública, 2=Empresa Privada).

O atributo comum às duas bases é a raiz do CNPJ. Observe que existem algumas empresas sem loja correspondente e algumas lojas sem empresa correspondente. O programa a seguir mostra como os diferentes tipos de junção envolvendo esses dois arquivos podem ser implementados.

```
#P16: junção de DataFrames
import pandas as pd

#(1)-importa os CSVs
df_emps = pd.read_csv('C:/CursoPython/emps.csv')
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

print('Empresas: '); print(df_emps)
print('-----------------------------------------------')
print('Lojas: '); print(df_lojas)
print('-----------------------------------------------')

#(2)-Combina os Arquivos

#2.1 INNER JOIN
c1 = pd.merge(df_emps, df_lojas, how='inner', on='raiz')
print('resultado do INNER JOIN: ')
print(c1)
print('-----------------------------------------------')

#2.2 LEFT JOIN
c2 = pd.merge(df_emps, df_lojas, how='left', on='raiz')
print('resultado do LEFT JOIN: ')
print(c2)
print('-----------------------------------------------')

#2.3 FULL OUTER JOIN
c3 = pd.merge(df_emps, df_lojas, how='outer', on='raiz')
print('resultado do FULL OUTER JOIN: ')
print(c3)
print('-----------------------------------------------')
```

```
>>> %Run p16_merge.py
Empresas:
       raiz               razao  natjur
0  11111111  UNIVERSIDADE ALPHA       1
1  22222222           PYTHONISTA       2
2  33333333               NPSERV       1
3  44444444          MERCADO XYZ       2
4  55555555     JAVANESE BRASIL       2
5  66666666               CAFEDB       2
6  88888888                  ABG       1
-----------------------------------------------
```

```
Lojas:
        raiz  sufixo        fantasia   po  salario  uf
0   22222222       1  PYTHONSHOP I     20    35000  SP
1   22222222       2  PYTHONSHOP II     5     4000  RJ
2   22222222       3  PYTHONSHOP III   15    31000  MG
3   33333333       1  NPSERV LOJA A     8     9200  MG
4   33333333       2  NPSERV LOJA B     0        0  MG
5   44444444       1  XYZ MATRIZ      112   250000  RJ
6   44444444       3  XYZ CENTRO      101   103900  SP
7   44444444       4  XYZ NORTE        48    60000  RJ
8   44444444       5  XYZ SUL          50    65000  RJ
9   44444444      10  XYZ LITORAL       1     2500  RJ
10  55555555       1  JAVASERV         40    40000  SP
11  66666666       1  CAFEDB           12    12900  RS
12  99999999       1  PANDAS KING       1     3000  MG
------------------------------------------------
resultado do INNER JOIN:
        raiz            razao  natjur  sufixo        fantasia   po  salario  uf
0   22222222       PYTHONISTA       2       1  PYTHONSHOP I     20    35000  SP
1   22222222       PYTHONISTA       2       2  PYTHONSHOP II     5     4000  RJ
2   22222222       PYTHONISTA       2       3  PYTHONSHOP III   15    31000  MG
3   33333333           NPSERV       1       1  NPSERV LOJA A     8     9200  MG
4   33333333           NPSERV       1       2  NPSERV LOJA B     0        0  MG
5   44444444      MERCADO XYZ       2       1  XYZ MATRIZ      112   250000  RJ
6   44444444      MERCADO XYZ       2       3  XYZ CENTRO      101   103900  SP
7   44444444      MERCADO XYZ       2       4  XYZ NORTE        48    60000  RJ
8   44444444      MERCADO XYZ       2       5  XYZ SUL          50    65000  RJ
9   44444444      MERCADO XYZ       2      10  XYZ LITORAL       1     2500  RJ
10  55555555  JAVANESE BRASIL       2       1  JAVASERV         40    40000  SP
11  66666666           CAFEDB       2       1  CAFEDB           12    12900  RS
------------------------------------------------
resultado do LEFT JOIN:
        raiz                razao  natjur  ...     po   salario   uf
0   11111111  UNIVERSIDADE ALPHA       1  ...    NaN       NaN  NaN
1   22222222          PYTHONISTA       2  ...   20.0   35000.0   SP
2   22222222          PYTHONISTA       2  ...    5.0    4000.0   RJ
3   22222222          PYTHONISTA       2  ...   15.0   31000.0   MG
4   33333333              NPSERV       1  ...    8.0    9200.0   MG
5   33333333              NPSERV       1  ...    0.0       0.0   MG
6   44444444         MERCADO XYZ       2  ...  112.0  250000.0   RJ
7   44444444         MERCADO XYZ       2  ...  101.0  103900.0   SP
8   44444444         MERCADO XYZ       2  ...   48.0   60000.0   RJ
9   44444444         MERCADO XYZ       2  ...   50.0   65000.0   RJ
10  44444444         MERCADO XYZ       2  ...    1.0    2500.0   RJ
11  55555555     JAVANESE BRASIL       2  ...   40.0   40000.0   SP
12  66666666               CAFEDB       2  ...   12.0   12900.0   RS
13  88888888                  ABG       1  ...    NaN       NaN  NaN

[14 rows x 8 columns]
------------------------------------------------
resultado do FULL OUTER JOIN:
        raiz                razao  natjur  ...     po   salario   uf
0   11111111  UNIVERSIDADE ALPHA     1.0  ...    NaN       NaN  NaN
1   22222222          PYTHONISTA     2.0  ...   20.0   35000.0   SP
2   22222222          PYTHONISTA     2.0  ...    5.0    4000.0   RJ
3   22222222          PYTHONISTA     2.0  ...   15.0   31000.0   MG
4   33333333              NPSERV     1.0  ...    8.0    9200.0   MG
5   33333333              NPSERV     1.0  ...    0.0       0.0   MG
6   44444444         MERCADO XYZ     2.0  ...  112.0  250000.0   RJ
```

```
7   44444444         MERCADO XYZ     2.0  ...  101.0  103900.0   SP
8   44444444         MERCADO XYZ     2.0  ...   48.0   60000.0   RJ
9   44444444         MERCADO XYZ     2.0  ...   50.0   65000.0   RJ
10  44444444         MERCADO XYZ     2.0  ...    1.0    2500.0   RJ
11  55555555     JAVANESE BRASIL     2.0  ...   40.0   40000.0   SP
12  66666666              CAFEDB     2.0  ...   12.0   12900.0   RS
13  88888888                 ABG     1.0  ...    NaN       NaN  NaN
14  99999999                 NaN     NaN  ...    1.0    3000.0   MG

[15 rows x 8 columns]
------------------------------------------------
```

4.8 Limpeza

Limpeza é a atividade de data wrangling que consiste em eliminar sujeira e informações irrelevantes de uma base de dados. O programa a seguir apresenta um exemplo simples de limpeza, onde a função de string `trim()` é aplicada para remover os espaços em branco à direita do atributo "fantasia".

```
#P17: Limpeza
import pandas as pd

#(1)-Importa a base de dados para um DataFrame
df_lojas = pd.read_csv('C:/CursoPython/lojas.csv')

#(2)-Exibe o nome fantasia original. Veja que o atributo foi importado com
#    espaços em branco à direita
print('fantasia - valores originalmente importados:')
print(df_lojas['fantasia'].values)

#(3)-Utiliza o apply para remover os espaços em branco
df_lojas['fantasia'] = df_lojas['fantasia'].apply(str.rstrip)

#(4)-Agora os espaços em branco não existem mais
print('-----------------------------------------')
print('fantasia - valores após a limpeza dos espaços em branco:')
print(df_lojas['fantasia'].values)
```

```
>> %Run p17_limpeza.py
fantasia - valores originalmente importados:
['PYTHONSHOP I  ' 'PYTHONSHOP II ' 'PYTHONSHOP III' 'NPSERV LOJA A '
 'NPSERV LOJA B ' 'XYZ MATRIZ    ' 'XYZ CENTRO    ' 'XYZ NORTE     '
 'XYZ SUL       ' 'XYZ LITORAL   ' 'JAVASERV      ' 'CAFEDB        '
 'PANDAS KING   ']
-----------------------------------------
fantasia - valores após a limpeza dos espaços em branco:
['PYTHONSHOP I' 'PYTHONSHOP II' 'PYTHONSHOP III' 'NPSERV LOJA A'
 'NPSERV LOJA B' 'XYZ MATRIZ' 'XYZ CENTRO' 'XYZ NORTE' 'XYZ SUL'
 'XYZ LITORAL' 'JAVASERV' 'CAFEDB' 'PANDAS KING']
>>>
```

4.9 Exercícios

Utilize a 'pandas' para realizar as seguintes tarefas de exploração, transformação e limpeza da base de dados 'flags.csv' (que faz parte do material de apoio deste curso):

(a) importar a base 'flags.csv' para um DataFrame chamado "df".

(b) imprimir o total de linhas e o total de colunas da base

(c) imprimir as linhas finais e as linhas iniciais.

(d) listar o nome de todos os atributos da base.

(e) determinar todos os valores distintos dos atributos “language” e “religion”.

(f) gerar uma tabulação que exiba o total de países que fala cada língua.

(g) listar o nome de todos os países que possuem as cores ‘azul’ e ‘branco’ em sua bandeira.

(h) remover o atributo “colours”.

(i) Criar um novo atributo chamado “pais_continental” a partir da discretização do atributo numérico “area”. O atributo “pais_continental” deverá ter o valor ‘S’, se area >= 1000; caso contrário, deverá ter o valor ‘N’.

(j) Transformar o atributo “zone” em 4 atributos binários e, em seguida, projetar o atributo “zone” e os 4 atributos binários criados.

(k) Obter o maior valor, o menor valor, a média e a mediana do atributo “area”.

(l) Gerar um boxplot para o atributo “population”.

Referências

Corrêa, E., Meu Primeiro Livro de Python, v.2.0.0., edubd, 2019.

Corrêa, E., Pandas Python: Data Wrangling para Ciência de Dados, Casa do Código, 2020.

Han, J., Kamber, M. and Pei, J., Data Mining: Concepts and Techniques, 3rd ed., Morgan Kaufmann, 2011.

Hand, D. J. (2020) "Data Science", In: Wiley StatsRef: Statistics Reference Online, Edited by N. Balakrishnan, T. Colton, B. Everitt, W. Piegorsch, F. Ruggeri, and J.L. Teugels, Wiley, USA. doi:10.1002/9781118445112.stat08150

Kandel, S., Heer, J., Plaisant, C., Kennedy, J., van Ham, F., Riche, N.H., Weaver, C., Lee, B., Brodbeck, D., and Buono, P. (2011). Research directions in data wrangling: Visualizations and transformations for usable and credible data. In: Information Visualization, 10(4), pages 271–288.

McKinney, W. Python for Data Analysis: Data Wrangling with Pandas, NumPy and IPython. 2nd ed., O'Reilly, 2017.

McKinney, W. et al. (2020). Pandas: Powerful python data analysis toolkit release 1.0.1. Disponível em: <https://pandas.pydata.org/>.

Numpy (2020). Numpy v1.19.dev0 manual. Disponível em: <https://numpy.org/devdocs/index.html>

Press, G. (2016). Cleaning big data: most time-consuming, least enjoyable data science task, survey says. Forbes. Disponível em: <https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#63947c656f63>.

Ruiz, A. (2017). The 80/20 data science dilemma. InfoWorld. Disponível em: <https://www.infoworld.com/article/3228245/the-80-20-data-science-dilemma.html>.

Witten, I., Frank, E., Hall, M. and Pal, C. Data Mining: Practical Machine Learning Tools and Techniques. 4th ed., Morgan Kaufmann, 2016.

Zhang, S., Zhang, C., and Yang, Q. (2003). Data preparation for data mining. In: Applied Artificial Intelligence, 17(5-6), pages 375–381.

Anexo 1 - Bônus

Esta seção apresenta um exemplo de utilização da 'pandas' em conjunto com a 'scikit-learn' em um processo de machine learning não-supervisionado.

A biblioteca scikit-learn é voltada para o desenvolvimento de projetos de Machine Learning em Python. Ela contém uma série de módulos que oferecem algoritmos a criação e avaliação de modelos de classificação, regressão e clustering (além de pacotes para a execução de tarefas de pré-processamento, como seleção de atributos e redução da dimensionalidade). Normalmente, todo programa que utiliza esta biblioteca faz o import de forma semelhante á mostrada abaixo:

```
from sklearn.modulo import algoritmo as apelido
```

Onde:

- **modulo** consiste em algum dos módulos disponibilizados pela scikit-learn. Normalmente eles são organizados de acordo com o tipo de tarefa a ser executada (classificação, clustering, avaliação de modelos, etc.) e, às vezes, também considerando uma determinada "família" (princípio matemático) de algoritmos. Por exemplo sklearn.naive_bayes é o módulo de algoritmos para classificação da "família" Naïve Bayes (classificação bayesiana).

- **algoritmo** indica o algortimo a ser selecionado dentro do módulo. Por exemplo, o módulo sklearn.naive_bayes oferece os algortimos "GaussianNB", "MultinomialNB" e "BernoulliNB" (são basicamente, três versões distintas do Naïve Bayes).

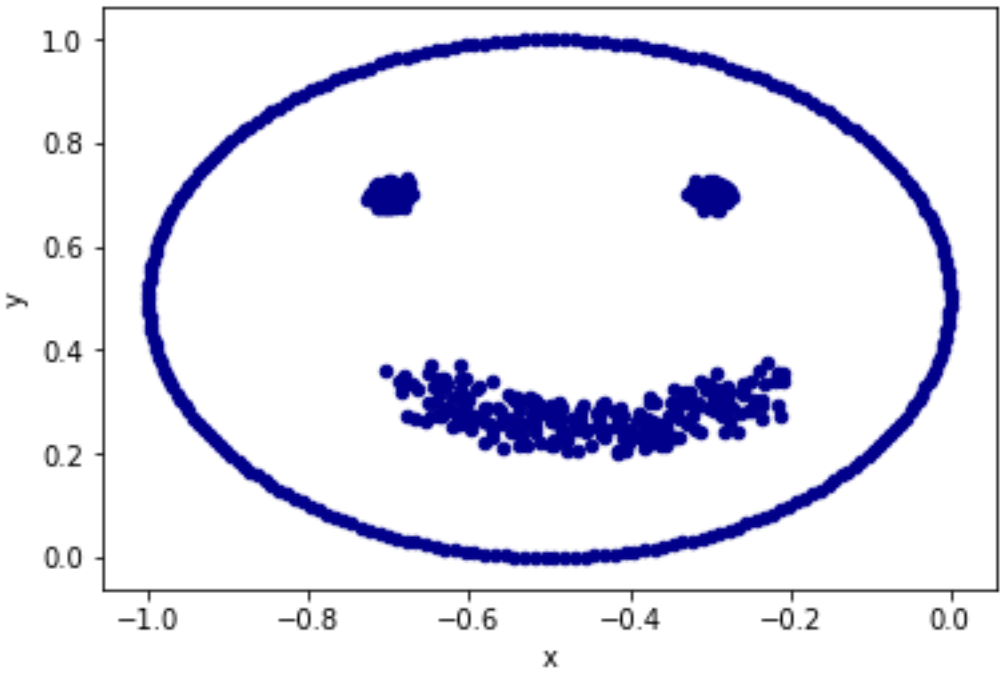
- **apelido** serve apenas para simplificar a referência ao algoritmo no corpo do programa. É a mesma coisa que fazemos ao apelidar a Numpy de "np".

Nesta seção, apresentamos um exemplo simples que envolve o utilização da scikit-learn para o conhecido algoritmo k-Means para aprendizado não supervisionado [Han et al., 2011; Witten et al., 2016]. Utilizaremos como teste, a base "SMILE.csv". Essa base possui apenas duas dimensões e seu conteúdo pode ser exibido em um gráfico de dispersão com a utilização do código da página a seguir. Veja que existem quatro clusters bem definidos: o contorno da face, olho esquerdo, olho direito e boca.

```
#P16: desenha smile
import pandas as pd

#(1)-importa o CSV
df_smile = pd.read_csv('C:/CursoPython/smile.csv')
df_smile.columns = ["x","y"]

#(2)-imprime as primeiras linhas (só para checar...)
print(df_smile.head())

#(3)-imprime um sumário das estatísticas de "x" e "y"
print(df_smile.describe())

#(4)-plota o gráfico
df_smile.plot(kind="scatter",x="x",y="y", c="darkblue")
```

```
>>> %Run p16_plota_smile.py
          x         y
```

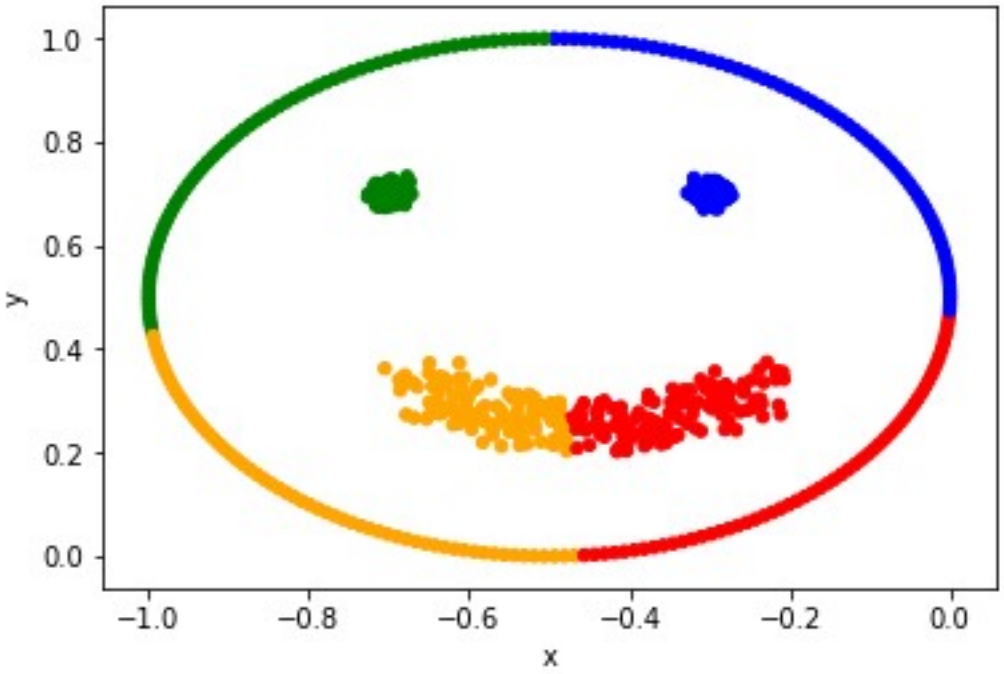
```
0 -0.288011  0.692245
1 -0.306319  0.708402
2 -0.304490  0.714163
3 -0.288526  0.693369
4 -0.294865  0.703397
                x           y
count  999.000000  999.000000
mean    -0.487231    0.544219
std      0.237229    0.248472
min     -1.000000    0.000039
25%     -0.695881    0.292812
50%     -0.465254    0.689553
75%     -0.297309    0.703789
max      0.000000    0.999961
```



A seguir mostra-se como utilizar o k-Means da scikit-learn, configurando k = 4. Após a execução do algoritmo, os clusters são exibidos com cores diferentes. As explicações sobre como carregar, configurar, executar e verificar os resultados do k-Means são apresentadas dentro do próprio corpo do programa.

```
#P17: kmeans
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans

#(1)-importa o CSV
df_smile = pd.read_csv('C:/CursoPython/smile.csv')
df_smile.columns = ["x","y"]

#(2)-Executa o kMeans sobre a base importada
print(df_smile.head())
```

```
#(3)-imprime um sumário das estatísticas de "x" e "y"
modelo = KMeans(n_clusters=4) #configura k=4

modelo.fit(df_smile) #executa o algoritmo

#retorna um vetor de inteiros com os ids dos clusters que foram
#associados a cada observação
#- como k=4, existem 4 clusters possíveis: 0, 1, 2 e 3
#- Ex.: grupos[0] = 2, sigifica que a primeira observação da base foi
# associada ao cluster 2
#- Sendo asisim, o tamanho (número de linhas) do vetor de clusters é
# igual ao da base de dados

grupos = modelo.labels_

#(4)-plota o gráfico
colormap = np.array(['red','green','blue','orange'])
df_smile.plot(kind="scatter",x="x",y="y", c=colormap[grupos])
```



O resultado ruim era esperado e se deve ao fato de o k-Means não ser um algoritmo adequado para trabalhar com fronteiras de decisão não-lineares. Como sugestão, experimente outros algoritmos de clustering da scikit-learn e tente encontrar algum que seja adequado para uma base com as características da "SMILE.csv".