

NÃO PODE FALTAR

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM P

Vanessa Cadan Scheffer

MÉTODOS PARA MANIPULAÇÃO DE DADOS

Além dos métodos para carregar e salvar dados, a biblioteca pandas possui métodos para a transformação dos dados e a extração de informação.



Fonte: Shutterstock.

(sistemas de armazenamento em cloud) ou em bancos de dados de diversas origens (e até mais), a biblioteca possui métodos capazes de fazer a leitura dos dados e carregar em um DataFrame.

Todos os métodos capazes de fazer a leitura dos dados estruturados possuem o prefixo `pd.read_XXX`, onde `pd` é o apelido dado no momento da importação da biblioteca e `XXX` é o restante da sintaxe do método. Além de fazer a leitura, a biblioteca possui diversos métodos capazes de escrever o DataFrame em um arquivo, em um banco ou ainda simplesmente copiar para a memória do sistema operacional. O Quadro 4.2, apresenta todos os métodos para leitura e escrita. Veja que são suportados tanto a leitura de arquivos de texto quanto binários e de bancos.

Quadro 4.2 - Métodos para leitura e escrita de dados estruturados

Tipo de dado	Descrição do dado	Método para leitura	Método para escrita
texto	<u>CSV</u>	<code>read_csv</code>	<code>to_csv</code>
texto	Fixed-Width texto File	<code>read_fwf</code>	<code>to_fwf</code>
texto	<u>JSON</u>	<code>read_json</code>	<code>to_json</code>

Tipo de dado	Descrição do dado	Método para leitura	Método para escrita
binário	<u>Feather Format</u>	read_feather	to_feather
binário	<u>Parquet Format</u>	read_parquet	to_parquet
binário	<u>ORC Format</u>	read_orc	to_orc
binário	<u>Msgpack</u>	read_msgpack	to_msgpack
binário	<u>Stata</u>	read_stata	to_stata
binário	<u>SAS</u>	read_sas	to_sas
binário	<u>SPSS</u>	read_spss	to_spss
binário	<u>Python Pickle Format</u>	read_pickle	to_pickle
SQL	<u>SQL</u>	read_sql	to_sql
SQL	<u>Google BigQuery</u>	read_gbq	to_gbq

Fonte: adaptado de <https://pandas.pydata.org/pandas-docs/stable/10min.html>

Dentre todos os possíveis métodos para leitura, nessa aula vamos ver a função read_csv e a função read_sql que contempla a função read_sql.

CSV (comma-separated values - valores separados por vírgula) é um formato de arquivo, nos quais os dados são separados por um delimitador. Nesse delimitador é uma vírgula (por isso o nome), mas na prática pode ser criado com qualquer delimitador, por exemplo, por pipe (|), dentre outros. Por ser um arquivo de texto, é fácil de ser lido por qualquer sistema, por isso se tornou tão democrático.

LEITURA DE JSON E CSV COM PANDAS

Agora que vamos começar a ver as implementações, vamos falar sobre a biblioteca, como já sabemos, só precisamos importar uma única biblioteca ou no script .py.

```
In [1]: import pandas as pd
```

A leitura de um arquivo JSON deve ser feita com o método: `pandas.read_json(path_or_buf=None, orient=None, typ='frame', convert_axes=None, convert_dates=True, keep_default_dates=True, precise_float=False, date_unit=None, encoding=None, lines=False, compression='infer')`. Os detalhes de cada parâmetro podem ser encontrados na documentação oficial: https://pandas.pydata.org/pandas-docs/stable/10min/05io.html#pandas.read_json. O único parâmetro que é obrigatório é o "path_or_buf", no qual deve ser passado um caminho para o arquivo.

In [2]:

```
pd.read_json("https://api.bcb.gov.br/dados/serie/  
formato=json").head()
```

Out[2]:

	data	valor
0	04/06/1986	0.065041
1	05/06/1986	0.067397
2	06/06/1986	0.066740
3	09/06/1986	0.068247
4	10/06/1986	0.067041

A leitura de um arquivo CSV deve ser feita com o método:

```
pandas.read_csv(filepath_or_buffer: Union[str, pathlib.Path,  
sep=',', delimiter=None, header='infer', names=None, index_col=None,  
usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True,  
engine=None, converters=None, true_values=None, false_values=None,  
skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None,  
keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True,  
parse_dates=False, infer_datetime_format=False, keep_date_col=False,  
date_parser=None, dayfirst=False, cache_dates=True, iterator=False,  
chunksize=None, compression='infer', thousands=None, decimal=',',  
lineterminator=None, quotechar='"', quoting=0, doublequote=True,  
escapechar=None, comment=None, encoding=None, dialect=None,  
error_bad_lines=True, warn_bad_lines=True, delim_whitespace=False,  
low_memory=True, memory_map=False, float_precision=None). São  
o que proporciona uma versatilidade incrível para esse método
```

que estão separados por vírgula. O parâmetro `header`, tem como valor `'infer'`, que significa que o método realiza a inferência para os nomes das colunas a partir da primeira linha de dados do arquivo.

Na entrada 3, estamos fazendo a leitura de uma fonte CSV, cujos dados estão separados por vírgula, logo não foi preciso especificar um delimitador.

In [3]:

```
pd.read_csv("https://people.sc.fsu.edu/~jburkardt
```

Out[3]:

	LatD	"LatM"	"LatS"	"NS"	"LonD"	"LonM"	"LonS"	"EW"	"City"	"State"
0	41	5	59	"N"	80	39	0	"W"	"Youngstown"	OH
1	42	52	48	"N"	97	23	23	"W"	"Yankton"	SD
2	46	35	59	"N"	120	30	36	"W"	"Yakima"	WA
3	42	16	12	"N"	71	48	0	"W"	"Worcester"	MA
4	43	37	48	"N"	89	46	11	"W"	"Wisconsin Dells"	WI

MANIPULAÇÃO DE DADOS COM PANDAS

Além de vários métodos para carregar e salvar os dados, a biblioteca `Pandas` possui uma diversidade de métodos para a transformação dos dados, permitindo a extração de informação para áreas de negócio. Nessa seção vamos conhecer alguns desses métodos.

O trabalho com dados: capturar os dados em suas origens, fazer a limpeza dos dados a fim de padronizá-los, aplicar técnicas estatísticas e aplicar algoritmos de machine/deep learning feito por engenheiros e cientistas de dados.

por exemplo, se você pretende fazer um financiamento ou investimento, precisa olhar a taxa Selic, pois ela influencia o pago ou ganho. "Selic é a sigla para Sistema Especial de Liquidação e Custódia, um programa totalmente virtual em que os títulos do Tesouro Nacional são comprados e vendidos diariamente por instituições financeiras" (NUBANK, p. 1, 2020). Quem decide, de fato, a taxa Selic é o Copom (Comitê de Política Monetária do Banco Central), que se reúne para determinar se a taxa aumenta ou diminui. Se a taxa está alta, os financiados podem ficar mais caros e o consumo se a taxa está mais baixa, então os financiamentos ficam mais baratos. Resumindo, quando a taxa Selic aumenta a economia de crédito, e quando ela abaixa a economia aquece, isso é preciso para controlar a inflação.

Agora que já conhecemos que a taxa Selic influencia, nossos próximos passos são analisar investimentos e até mesmo o que compramos no mercado, vamos coletar as informações disponibilizadas pelo governo e fazer algumas análises. Vamos começar pela etapa de extração e transformação dos dados.

ETAPA DE CAPTURA E TRANSFORMAÇÃO/ PADRONIZAÇÃO DOS DADOS

A extração dos dados pode ser realizada por meio do método `read_html()` guardando em um DataFrame (DF) pandas. Ao carregar os dados, podemos visualizar quantas linhas e colunas, bem como os tipos de dados.

linhas, então não existem valores faltantes. Quanto ao tipo de object, ou seja, são todos do tipo strings ou existe mistura de float.

In [4]:

```
df_selic =  
pd.read_json("https://api.bcb.gov.br/dados/serie/  
formato=json")
```

```
print(df_selic.info())
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 8552 entries, 0 to 8551  
Data columns (total 2 columns):  
data      8552 non-null object  
valor     8552 non-null float64  
dtypes: float64(1), object(1)  
memory usage: 133.7+ KB  
None
```


REMOVER LINHAS DUPLICADAS

Para o carregamento de uma base de dados, um dos primeiros passos que devemos fazer é remover os dados duplicados. Certamente, que isso depende da área de negócio e do problema a ser resolvido. Por exemplo, se queremos manter o registro da compra atual, ou queremos manter a primeira compra. Um DataFrame da biblioteca pandas possui o método `meu_df.drop_duplicates()` que permite fazer essa remoção de duplicatas.

Observe a entrada 5 da linha de código a seguir, usamos o método `drop_duplicates(keep='last')` para remover as linhas duplicadas, pedindo para manter o último registro (keep='last'). Se o parâmetro `inplace=True`, estamos fazendo com que a transformação seja feita no DataFrame, na prática estamos sobrescrevendo o objeto na memória. **inplace não seja passado, a transformação é aplicada, mas o DataFrame original não seja, o DF continua da mesma forma anterior a transformação.** Um parâmetro interessante do método é o `subset`, que permite que a remoção de dados duplicado seja feita com base em uma ou mais colunas.

In [5]:

```
df_selic.drop_duplicates(keep='last', inplace=True)
```

CRIAR NOVAS COLUNAS

tipo de dados das datas, embora cada valor seja do tipo "date" ainda obtemos uma coluna object, para que de fato, a biblioteca um tipo data, vamos ter que utilizar o método da própria biblioteca para a conversão.

In [6]:

```
from datetime import date
from datetime import datetime as dt

data_extracao = date.today()

df_selic['data_extracao'] = data_extracao
df_selic['responsavel'] = "Autora"

print(df_selic.info())
df_selic.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8552 entries, 0 to 8551
Data columns (total 4 columns):
data                8552 non-null object
valor               8552 non-null float64
data_extracao       8552 non-null object
responsavel         8552 non-null object
dtypes: float64(1), object(3)
memory usage: 334.1+ KB
None
```

Out[6]:

	data	valor	data_extracao	responsavel
0	04/06/1986	0.065041	2020-07-19	Autora
1	05/06/1986	0.067397	2020-07-19	Autora
2	06/06/1986	0.066740	2020-07-19	Autora
3	09/06/1986	0.068247	2020-07-19	Autora
4	10/06/1986	0.067041	2020-07-19	Autora

guardamos dentro da própria coluna, dessa forma os valores s
linha 1, usamos a notação `pd.to_datetime()`, porque é um méto
não do DF.

Na entrada 7 (linha 1), o método recebe a coluna com os valores
(`df_selic['data']`) e um segundo parâmetro, indicando que no
(antes da conversão) o dia está primeiro (`dayfirst=True`). Em se
como a coluna "data_extracao" foi criada com o método `today`
correto para a conversão. Nessa conversão usamos o método
transforma os dados de uma coluna (que é uma Series) em um
nesse caso, o tipo `datetime` especificado. Com `astype()` podemo
valores das colunas, por exemplo, transformando todos em flo
outro tipo. Veja que agora, ao usar o método `info()`, temos que
do tipo `datetime` (`datetime` da biblioteca `pandas`). O formato re
dia é um padrão do `datetime64[ns]`, que segue o padrão intern
ano vem primeiro, seguido do mês e por último o dia. Poderían
para transformar o traço em barra (/), mas aí o resultado seria
datas.

In [7]:

```
df_selic['data'] = pd.to_datetime(df_selic['data'], dayfirst=True)
df_selic['data_extracao'] = df_selic['data_extracao'].astype('datetime64[ns]')

print(df_selic.info())
```

	data	valor	data_extracao	responsavel
0	1986-06-04	0.065041	2020-07-19	Autora
1	1986-06-05	0.067397	2020-07-19	Autora
2	1986-06-06	0.066740	2020-07-19	Autora
3	1986-06-09	0.068247	2020-07-19	Autora
4	1986-06-10	0.067041	2020-07-19	Autora

■ SERIES.STR

Muitas vezes precisamos padronizar os valores em colunas, por exemplo, queremos ter certeza que a coluna "responsável" possui todas as células padronizadas em letras maiúsculas. Quando selecionamos um elemento de uma coluna, sabemos que o resultado é uma Series e esse objeto tem um método `str` que permite aplicar as funções de string para todos os valores da Series.

Observe o trecho de código seguir, selecionamos a coluna `responsavel` usando o recurso `str` e aplicamos o método `upper()`. Dessa forma, a biblioteca `pandas` "entende" que queremos converter todos os valores dessa coluna para letras maiúsculas. Como atribuímos o resultado na própria coluna, o valor original é substituído.

In [8]:

```
df_selic['responsavel'] = df_selic['responsavel'].str.upper()

df_selic.head()
```

Out[8]:

	data	valor	data_extracao	responsavel
0	1986-06-04	0.065041	2020-07-19	AUTORA

■ MÉTODO SORT_VALUES()

No código a seguir, estamos usando o método `sort_values()` que ordena o DF, de acordo com os valores de uma coluna. Esse método é chamado assim a notação `meu_df.metodo()`. Utilizamos três parâmetros do método: o primeiro informando qual coluna deve ser usada para ordenar, o segundo que seja feito em ordem decrescente (do maior para o menor) e o terceiro (`inplace=True`) significa que queremos modificar o próprio objeto, estamos sobrescrevendo o DF.

In [9]:

```
df_selic.sort_values(by='data', ascending=False, inplace=True)
df_selic.head()
```

Out[9]:

	data	valor	data_extracao	responsavel
8551	2020-07-17	0.008442	2020-07-19	AUTORA
8550	2020-07-16	0.008442	2020-07-19	AUTORA
8549	2020-07-15	0.008442	2020-07-19	AUTORA
8548	2020-07-14	0.008442	2020-07-19	AUTORA
8547	2020-07-13	0.008442	2020-07-19	AUTORA

■ MÉTODO RESET_INDEX() E SET_INDEX()

Ao fazermos a ordenação dos dados com o método `sort_values()`, os índices dos cinco primeiros registros é 8551, 8550...85XX. Nenhum deles é o índice original do DataFrame.

em uma nova coluna e inplace, informa para gravar as alterações no mesmo objeto. Veja na saída que agora os cinco primeiros registros, por exemplo, são os de 2020-07-13 a 2020-07-17.

In [10]:

```
df_selic.reset_index(drop=True, inplace=True)

df_selic.head()
```

Out[10]:

	data	valor	data_extracao	responsavel
0	2020-07-17	0.008442	2020-07-19	AUTORA
1	2020-07-16	0.008442	2020-07-19	AUTORA
2	2020-07-15	0.008442	2020-07-19	AUTORA
3	2020-07-14	0.008442	2020-07-19	AUTORA
4	2020-07-13	0.008442	2020-07-19	AUTORA

Durante a transformação dos dados, pode ser necessário definir uma nova sequência de índices para os dados, ao invés de usar o range numérico. Essa transformação pode ser feita usando o método `meu_df.set_index()`. O método permite definir novos valores usando uma coluna já existente ou então passando uma sequência de valores igual a quantidade de linhas.

Observe os códigos nas entradas 11 e 12. Na entrada 11, estamos criando uma nova lista que usa os índices do DF, adicionando um prefixo para cada elemento. Nas entradas 12 são impressos os cinco primeiros itens da nova lista. Na entrada 13, estamos definindo o novo índice com base na lista criada. Veja que o parâmetro `new_index` recebe como parâmetro uma lista de lista e o segundo parâmetro

In [12]:

```
df_selic.set_index(keys=[lista_novo_indice], inplace=True)

df_selic.head()
```

Out[12]:

	data	valor	data_extracao	responsavel
selic_0	2020-07-17	0.008442	2020-07-19	AUTORA
selic_1	2020-07-16	0.008442	2020-07-19	AUTORA
selic_2	2020-07-15	0.008442	2020-07-19	AUTORA
selic_3	2020-07-14	0.008442	2020-07-19	AUTORA
selic_4	2020-07-13	0.008442	2020-07-19	AUTORA

Ao especificar um índice com valor conceitual, podemos usar a função `idxmax()` para descobrir qual o índice do menor e do maior de `valor` nos exemplos a seguir.

In [13]:

```
print(df_selic['valor'].idxmin())
print(df_selic['valor'].idxmax())
```

selic_7606

selic_7623

ETAPA DE EXTRAÇÃO DE INFORMAÇÕES

Agora que já fizemos as transformações que gostaríamos, podemos fazer a extração de informações.

FILTROS COM LOC

Um dos recursos mais utilizados por equipes das áreas de dados é o método `loc`.

DataFrames da biblioteca pandas possuem uma propriedade `loc` (pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html).

Essa propriedade permite acessar um conjunto de linhas (filtrado pelo índice ou por um vetor booleano (vetor de True ou False)).

Vamos começar explorando o filtro pelos índices. Entrada 14 e 15 (`loc`), o registro que possui índice 'selic_0', como resultado obtido na entrada 15, foram filtrados três registros, para isso foi necessário criar um DataFrame contendo os índices, como resultado obtivemos um novo DF. Na entrada 16 fizemos um fatiamento (`slice`), procurando um intervalo de índices.

```
In [14]: df_selic.loc['selic_0']
```

```
Out[14]: data          2020-07-17 00:00:00
valor              0.008442
data_extracao      2020-07-19 00:00:00
responsavel        AUTORA
Name: selic_0, dtype: object
```

```
In [15]: df_selic.loc[['selic_0', 'selic_4', 'selic_200']]
```

```
Out[15]:
```

	data	valor	data_extracao	responsavel
selic_0	2020-07-17	0.008442	2020-07-19	AUTORA
selic_4	2020-07-13	0.008442	2020-07-19	AUTORA
selic_200	2019-09-30	0.020872	2020-07-19	AUTORA

```
In [16]: df_selic.loc[:'selic_5']
```

```
Out[16]:
```

	data	valor	data_extracao	responsavel
selic_0	2020-07-17	0.008442	2020-07-19	AUTORA
selic_1	2020-07-16	0.008442	2020-07-19	AUTORA
selic_2	2020-07-15	0.008442	2020-07-19	AUTORA

- `df_selic.loc[['selic_0', 'selic_4', 'selic_200']]['valor']`
- `df_selic.loc[['selic_0', 'selic_4', 'selic_200']][['valor', 'data_extracao']]`

In [17]: `df_selic.loc[['selic_0', 'selic_4', 'selic_200']],`

Out[17]:

selic_0	0.008442
selic_4	0.008442
selic_200	0.020872
Name: valor, dtype: float64	

In [18]: `df_selic.loc[['selic_0', 'selic_4', 'selic_200'], 'data_extracao']]`

Out[18]:

	valor	data_extracao
selic_0	0.008442	2020-07-19
selic_4	0.008442	2020-07-19
selic_200	0.020872	2020-07-19

Antes de vermos a criação de filtros para o `loc` com condições, vamos mencionar que existe também a propriedade `iloc`, a qual filtra considerando a posição que ocupam no objeto. Veja no exemplo usando o **`iloc`** para filtrar os 5 primeiros registros, usando a mesma fatiamento de listas. Essa propriedade não possui a opção de filtrar por colunas. Veja um exemplo a seguir.

In [19]: `df_selic.iloc[:5]`

Out[19]:

	data	valor	data_extracao	responsavel
selic_0	2020-07-17	0.008442	2020-07-19	AUTORA
selic_1	2020-07-16	0.008442	2020-07-19	AUTORA

Observe a partir da entrada 20 a seguir. Estamos utilizando um relacional, para testar se os valores da coluna 'valor', são menores que 0.01. Armazenamos o resultado em uma variável chamada teste. Veja que teste é uma Series, e na linha 4 teste[:5], estamos imprimindo os primeiros resultados do teste lógico.

In [20]:

```
teste = df_selic['valor'] < 0.01

print(type(teste))
teste[:5]
```

```
<class 'pandas.core.series.Series'>
```

Out[20]:

```
selic_0    True
selic_1    True
selic_2    True
selic_3    True
selic_4    True
Name: valor, dtype: bool
```

No código, entrada 21 a seguir, realizamos mais um teste lógico para verificar se a taxa é do ano de 2020. Para isso, utilizamos o método to_datetime para converter a string para data e então fazer a comparação.

In [21]:

```
teste2 = df_selic['data'] >= pd.to_datetime('2020-01-01')

print(type(teste2))
teste2[:5]
```

```
<class 'pandas.core.series.Series'>
```

Out[21]:

```
selic_0    True
selic_1    True
```

parênteses, senão ocorre um erro. Observe o código a seguir, na linha seguinte, temos a construção de dois novos testes, o primeiro usando a operação AND e o segundo usando OR.

In [22]:

```
teste3 = (df_selic['valor'] < 0.01) & (df_selic['data'] > pd.to_datetime('2020-01-01'))

teste4 = (df_selic['valor'] < 0.01) | (df_selic['data'] > pd.to_datetime('2020-01-01'))

print("Resultado do AND:\n")
print(teste3[:3])

print("Resultado do OR:\n")
print(teste4[:3])
```

Resultado do AND:

```
selic_0    True
selic_1    True
selic_2    True
dtype: bool
```

Resultado do OR:

```
selic_0    True
selic_1    True
selic_2    True
dtype: bool
```

Agora que já sabemos criar as condições, basta aplicá-las no DataFrame para criar o filtro. A construção é feita passando a condição para a propriedade `loc` do código a seguir. Na linha 1 estamos criando a condição do filtro (uma `Series` booleana) e na entrada 23, passamos como parâmetro o DataFrame original.

	data	valor	data_extracao	responsavel
selic_5	2020-07-10	0.008442	2020-07-19	AUTORA
selic_6	2020-07-09	0.008442	2020-07-19	AUTORA
selic_7	2020-07-08	0.008442	2020-07-19	AUTORA
selic_8	2020-07-07	0.008442	2020-07-19	AUTORA
selic_9	2020-07-06	0.008442	2020-07-19	AUTORA
selic_10	2020-07-03	0.008442	2020-07-19	AUTORA
selic_11	2020-07-02	0.008442	2020-07-19	AUTORA
selic_12	2020-07-01	0.008442	2020-07-19	AUTORA
selic_13	2020-06-30	0.008442	2020-07-19	AUTORA
selic_14	2020-06-29	0.008442	2020-07-19	AUTORA
selic_15	2020-06-26	0.008442	2020-07-19	AUTORA
selic_16	2020-06-25	0.008442	2020-07-19	AUTORA
selic_17	2020-06-24	0.008442	2020-07-19	AUTORA
selic_18	2020-06-23	0.008442	2020-07-19	AUTORA
selic_19	2020-06-22	0.008442	2020-07-19	AUTORA
selic_20	2020-06-19	0.008442	2020-07-19	AUTORA
selic_21	2020-06-18	0.008442	2020-07-19	AUTORA
selic_7606	1990-03-16	0.000000	2020-07-19	AUTORA
selic_7607	1990-03-15	0.000000	2020-07-19	AUTORA
selic_7608	1990-03-14	0.000000	2020-07-19	AUTORA

Na entrada 23, criamos primeiro a condição, guardamos na variável `df_selic` e aplicamos, mas poderíamos ter passado a condição direta: `df_selic.loc[df_selic['valor'] < 0.01]`. Cabe ao desenvolvedor escolher a sintaxe que se sente mais a vontade. Nesse livro vamos adotar a sintaxe `df_selic.loc[df_selic['valor'] < 0.01]` e guardar em variáveis por questões didáticas e de legibilidade.

Na entrada 24 (linha 1), criamos uma condição para exibir o resultado apenas do mês de janeiro de 2020. Primeiro criamos duas variáveis para armazenar as datas, na linha 4 criamos o filtro e na linha 6 o aplicativo

In [24]:

```
data1 = pd.to_datetime('2020-01-01')
data2 = pd.to_datetime('2020-01-31')

filtro_janeiro_2020 = (df_selic['data'] >= data1)
(df_selic['data'] <= data2)

df_janeiro_2020 = df_selic.loc[filtro_janeiro_2020]
df_janeiro_2020.head()
```

Out[24]:

	data	valor	data_extracao	responsavel
selic_114	2020-01-31	0.017089	2020-07-19	AUTORA
selic_115	2020-01-30	0.017089	2020-07-19	AUTORA
selic_116	2020-01-29	0.017089	2020-07-19	AUTORA
selic_117	2020-01-28	0.017089	2020-07-19	AUTORA
selic_118	2020-01-27	0.017089	2020-07-19	AUTORA

Vamos criar mais um filtro e um novo DF para que possamos ver os dados de janeiro de 2019. No código a seguir, vamos criar um novo DF contendo a data de início e fim do mês de janeiro do ano de 2019.

In [25]:

```
data1 = pd.to_datetime('2019-01-01')
data2 = pd.to_datetime('2019-01-31')

filtro_janeiro_2019 = (df_selic['data'] >= data1)
(df_selic['data'] <= data2)

df_janeiro_2019 = df_selic.loc[filtro_janeiro_2019]
df_janeiro_2019.head()
```

Out[25]:

	data	valor	data_extracao	responsavel
selic_367	2019-01-31	0.02462	2020-07-19	AUTORA
selic_368	2019-01-30	0.02462	2020-07-19	AUTORA
selic_369	2019-01-29	0.02462	2020-07-19	AUTORA
selic_370	2019-01-28	0.02462	2020-07-19	AUTORA

In [26]:

```
print('Mínimo geral = ', df_selic['valor'].min())
print('Mínimo janeiro de 2019 = ', df_janeiro_2019['valor'].min())
print('Mínimo janeiro de 2020 = ', df_janeiro_2020['valor'].min())
print('\n')

print('Máximo geral = ', df_selic['valor'].max())
print('Máximo janeiro de 2019 = ', df_janeiro_2019['valor'].max())
print('Máximo janeiro de 2020 = ', df_janeiro_2020['valor'].max())
print('\n')

print('Média geral = ', df_selic['valor'].mean())
print('Média janeiro de 2019 = ', df_janeiro_2019['valor'].mean())
print('Média janeiro de 2020 = ', df_janeiro_2020['valor'].mean())
print('\n')
```

```
Mínimo geral = 0.0
Mínimo janeiro de 2019 = 0.02462
Mínimo janeiro de 2020 = 0.017089
```

```
Máximo geral = 3.626
Máximo janeiro de 2019 = 0.02462
Máximo janeiro de 2020 = 0.017089
```

```
Média geral = 0.2863543465855944
Média janeiro de 2019 = 0.024619999999999999
Média janeiro de 2020 = 0.017089000000000003
```

Veja como os filtros permitem começar a tirar respostas para a pergunta: "A taxa de juros foi superior em janeiro de 2019 ou em janeiro de 2020?". No ano de 2019 tanto a mínima quanto a máxima foram superiores a de 2020. A máxima geral é bem superior a máxima desses meses, a média geral, que é bem superior, ou seja, nesses meses a taxa de juros foi inferior à média geral, sendo que em janeiro de 2020 foi ainda pior.



Poderíamos extrair diversas outras informações dos dados. To parte do cotidiado nos engenheiros, cientistas e analistas de d engenheiros de dados mais focados na preparação e disponibi os cientistas focados em responder questões de negócio, inclu modelos de machine learning e deep learning e os analistas, ta a perguntas de negócios e apresentando resultados. Se você g fizemos quem sabe não deva investir mais nessa área?!

SAIBA MAIS

Existem diversos portais que disponibilizam dados, por e kaggle, os portais brasileiros <https://www.portaldatransp> <https://www.dados.gov.br/dataset>, o portal <https://archive> <https://vincentarelbundock.github.io/Rda> [assets.php](https://vincentarelbundock.github.io/Rda), ou o <https://vincentarelbundock.github.io/Rda> [ml](https://vincentarelbundock.github.io/Rda). Enfim, são inúmeros os repositórios que podem ser

- `pandas.read_sql(sql, con, index_col=None, coerce_float=True, parse_dates=None, columns=None, chunksize=None)`
- `pandas.read_sql_query(sql, con, index_col=None, coerce_float=True, params=None, parse_dates=None, chunksize=None)`

O mínimo de parâmetros que ambos métodos exigem é a instrução de conexão com um banco de dados (`con`). A conexão com o banco pode ser feita usando uma outra biblioteca, por exemplo, `sqlalchemy` (para diversos bancos), `pyodbc` para SQL Server, `cx_Oracle` para Oracle e `psycopg2` para Postgresql, dentre outras. Seguindo as recomendações da PEP 249, as bibliotecas precisam fornecer um método "connect", o qual retorna a conexão. A sintaxe da string de conexão depende da biblioteca usada para dados.

A seguir apresentamos de maneira didática dois exemplos, o primeiro com conexão com um banco postgresql e outro com um banco mysql. A diferença entre eles é a importação da biblioteca específica e a criação da instância. Dessa forma, ao estabelecer conexão com um banco de dados, basta criar a instância em uma variável, basta passá-la como parâmetro do método da biblioteca pandas.

```
import psycopg2
```



```
conn = psycopg2.connect(conn_str)
```

```
query = "select * from XXX.YYYY"
```

```
df = pd.read_sql(query, conn)import mysql.connector
```

```
host = 'XXXXX'
```

```
port = 'XXXXX'
```

```
database = 'XXXXX'
```

```
username = 'XXXXX'
```

```
password = 'XXXXX'
```

```
conn_str = fr"host={host}, user={username}, passwd={password}  
{database}"
```

```
conn = mysql.connector.connect(host="localhost", user="root",  
database="bd")
```

```
query = "select * from XXX.YYYY"
```

```
df = pd.read_sql(query, conn)
```

REFERÊNCIAS E LINKS ÚTEIS

Kaggle. Titanic: Machine Learning from Disaster. Disponível em [e.com/c/titanic](https://www.kaggle.com/c/titanic)). Acesso em: 20 jun. 2020.

Pandas Team. pandas documentation. Disponível em: <https://pandas-docs/stable/index.html>. Acesso em: 17 jun. 2020.

Pandas Team. Series. Disponível em: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html>. Acesso em: 17 jun. 2020.