**Introdução da aula**



Qual é o foco da aula?

Nesta aula, vamos assimilar que lém de vários métodos para carregar e salvar os dados, a biblioteca pandas possui uma diversidade de métodos para a transformação dos dados e a extração de informação para áreas de negócio

Objetivos gerais de aprendizagem

Ao longo desta aula, você irá:

* definir a manipulação de dados em panda;
* examinar as series.str;
* demonstrar os filtros com testes booleanos.

**Situação-problema**

Como desenvolvedor em uma empresa de consultoria de software, você foi alocado em um projeto para uma empresa de geração de energia. Essa empresa tem interesse em criar uma solução que acompanhe as exportações de etanol no Brasil. Esse tipo de informação está disponível no [site](http://www.dados.gov.br/dataset)do governo brasileiro, em formatos CSV, JSON, dentre outros.

No [endereço](http://www.dados.gov.br/dataset/importacoes-e-exportacoes-de-etanol)é possível encontrar várias bases de dados (datasets), contendo informações de importação e exportação de etanol. O cliente está interessado em obter informações sobre a Exportação Etano Hidratado (barris equivalentes de petróleo) 2012-2020, cujo endereço [é](http://www.dados.gov.br/dataset/importacoes-e-exportacoes-de-etanol/resource/ca6a2afe-def5-4986-babc-b5e9875d39a5). Para a análise será necessário fazer o download do arquivo.

O cliente deseja uma solução que extraia as seguintes informações:

* Em cada ano, qual o menor e o maior valor arrecadado da exportação?
* Considerando o período de 2012 a 2019, qual a média mensal de arrecadamento com a exportação.
* Considerando o período de 2012 a 2019, qual ano teve o menor arrecadamento? E o menor?

Como parte das informações técnicas sobre o arquivo, foi lhe informado que se trata de um arquivo delimitado CSV, cujo separador de campos é ponto-e-vírgula e a codificação do arquivo está em ISO-8859-1. Como podemos obter o arquivo? Como podemos extrair essas informações usando a linguagem Python? Serão necessários transformações nos dados para obtermos as informações solicitadas?

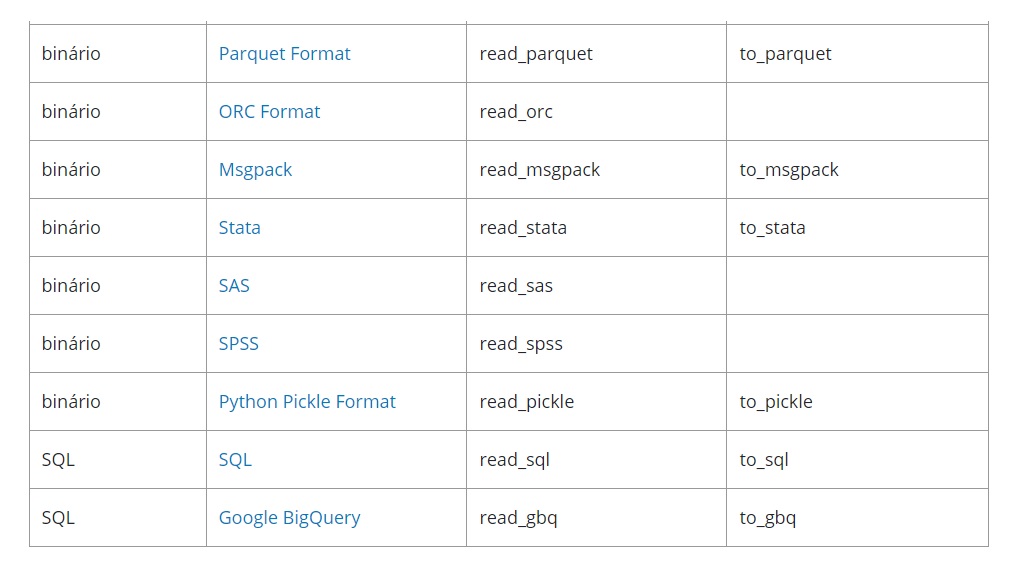
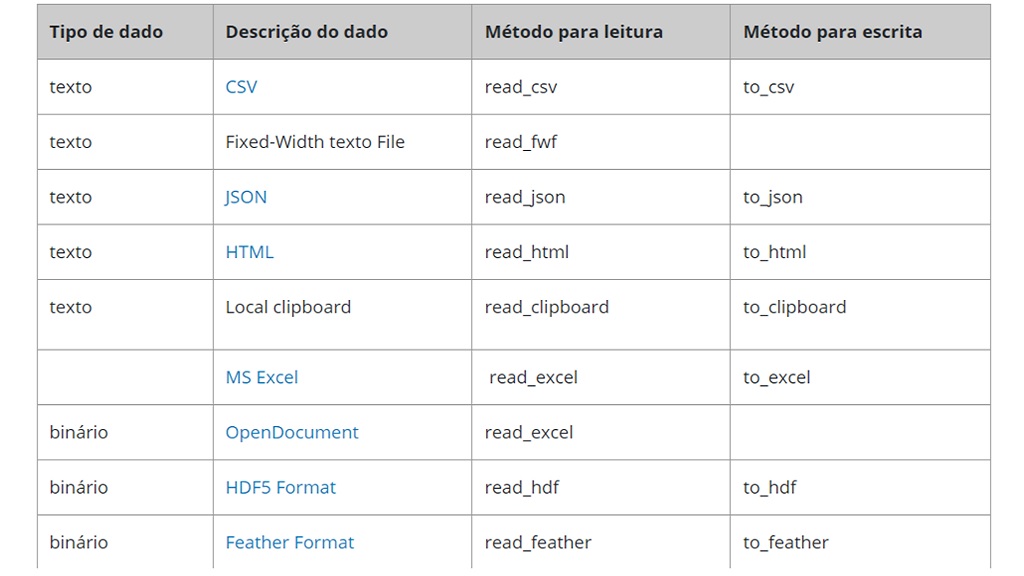
B

**Métodos para leitura e escrita da biblioteca pandas**



A biblioteca pandas foi desenvolvida para trabalhar com dados estruturados, ou seja, dados dispostos em linhas e colunas. Os dados podem estar gravados em arquivos, em páginas web, em APIs, em outros softwares, em object stores (sistemas de armazenamento em cloud) ou em bancos de dados. Para todas essas 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 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 área de transferência do sistema operacional. O quadro abaixo, apresenta todos os métodos de leitura e escrita. Veja que são suportados tanto a leitura de arquivos de texto, como binários e de bancos.

Métodos para leitura e escrita de dados estruturados. Fonte: adaptado de https://pandas.pydata.org/pandas-docs/stable/user\_guide/io.html

Dentre todos os possíveis métodos para leitura, nessa aula vamos estudar o read\_json, o read\_csv e a função read\_sql, que contempla a função read\_sql\_query.

**JSON**(JavaScript Object Notation - Notação de Objetos JavaScript) é uma formatação leve de troca de dados e independente de linguagem de programação. Os dados nesse formato podem ser encontrados como uma coleção de pares chave/valor ou uma lista ordenada de [valores](https://www.json.org/json-pt.html). Uma chave pode conter uma outra estrutura interna, criando um arquivo com multiníveis. Nessa aula vamos estudar somente a leitura de arquivos com um único nível, ou seja, para uma chave há somente um valor.

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

**Leitura de JSON e CVS com panda**s - Agora que vamos começar a ver as implementações, vamos fazer a importação da biblioteca, como já sabemos, só precisamos importar uma única vez no notebook ou no script .py.

In [1]: import pandas as pd

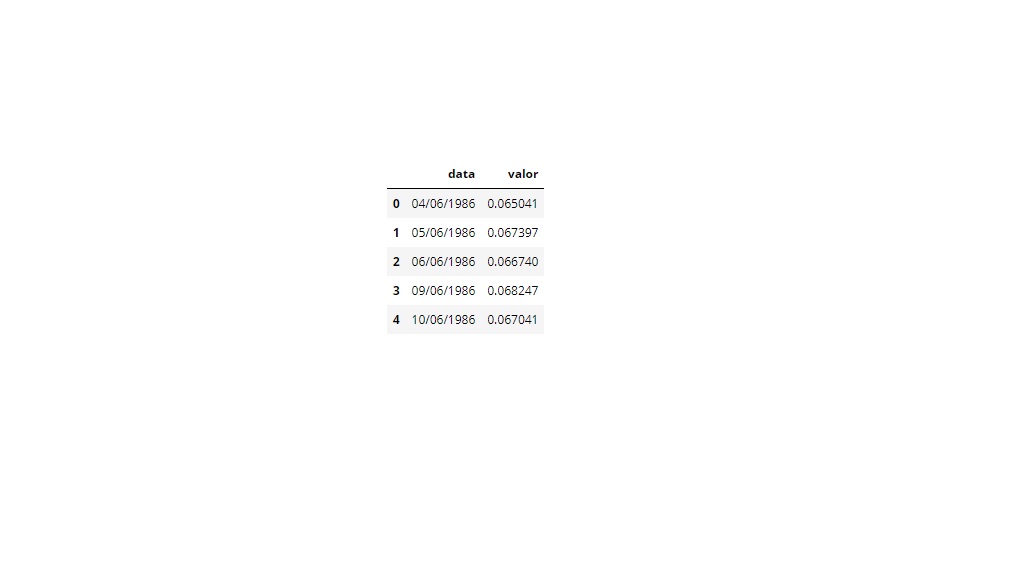
A leitura de um arquivo JSON deve ser feita com o método: pandas.read\_json(path\_or\_buf=None, orient=None, typ='frame', dtype=None, convert\_axes=None, convert\_dates=True, keep\_default\_dates=True, numpy=False, precise\_float=False, date\_unit=None, encoding=None, lines=False, chunksize=None, compression='infer'). Os detalhes de cada parâmetro podem ser consultados na documentação [oficial](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_json.html). O único parâmetro que é obrigatório para se carregar os dados é o "path\_or\_buf", no qual deve ser passado um caminho para o arquivo ou um "arquivo como objeto" que é um arquivo lido com a função open(), por exemplo.

Na entrada 2, estamos usando o método read\_json para carregar dados de uma API. Veja que estamos passando o caminho para o método. Nessa fonte de dados, são encontradas a taxa selic de cada dia.

In [2]:

pd**.**read\_json("https://api.bcb.gov.br/dados/serie/bcdata.sgs.11/dados?formato=json")**.**head()

Out [2]:



A leitura de um arquivo CSV deve ser feita com o método: pandas.read\_csv(filepath\_or\_buffer: Union[str, pathlib.Path, IO[~ AnyStr]], sep=',', delimiter=None, header='infer', names=None, index\_col=None, usecols=None, squeeze=False, prefix=None, mangle\_dupe\_cols=True, dtype=None, engine=None, converters=None, true\_values=None, false\_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na\_values=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: str = '.', 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 muitos parâmetros, o que proporciona uma versatilidade incrível para esse método. Os detalhes de cada parâmetro podem ser consultados na documentação [oficial](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html). Vamos comentar alguns.

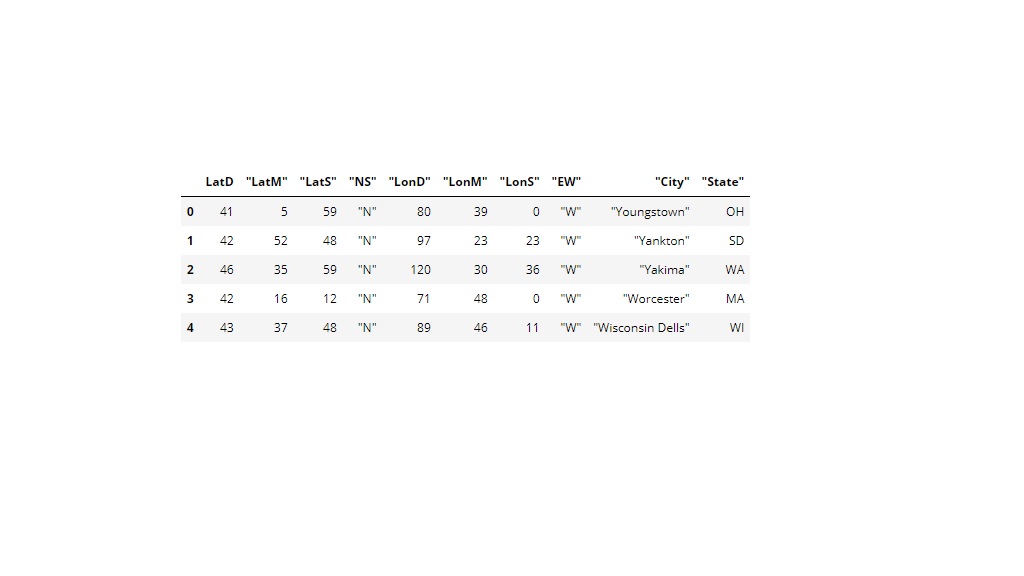
Para realizar o carregamento dos dados, é necessário incluir o caminho(diretório), portanto o parâmetro "filepath\_or\_buffer" é obrigatório. Outro parâmetro que é importante para a leitura desse arquivo é o sep ou delimiter (ambos fazem a mesma coisa), veja que sep, por padrão possui o valor ',', ou seja, caso não seja especificado nenhum valor, então o método fará a leitura dos dados considerando que estão separados por vírgula. O parâmetro header, tem como valor padrão '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 campos são separados por vírgula, logo não foi preciso especificar um delimitador diferente.

In [3]:

pd**.**read\_csv("https://people.sc.fsu.edu/~jburkardt/data/csv/cities.csv")**.**head()

Out [3]:



**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 e a extração de informação para áreas de negócio. Nesta aula vamos conhecer alguns deles.

O trabalho com dados: capturar os dados em suas origens, fazer transformações nos dados a fim de padronizá-los, aplicar técnicas estatísticas clássicas ou algoritmos de machine/deep learning feito por engenheiros e cientistas de dados. Cada profissional atuando em uma parte específica, dependendo da organização da empresa. Em todo esse trabalho é comum fazer a divisão em duas etapas: (i) captura e transformação/padronização dos dados, (ii) extração de informações.

\_\_\_\_\_\_\_

**📝 Exemplificando**

Para entender como essas duas etapas acontecem no dia a dia das empresas, vamos utilizar os dados da taxa Selic. Selic, ou taxa [Selic](https://blog.nubank.com.br/taxa-selic/)é a referência base de juros da economia brasileira. O valor da Selic influencia todo o setor financeiro do Brasil, por exemplo, se você pretende fazer um financiamento ou um investimento, precisa olhar a taxa Selic, pois ela influenciará o valor a ser 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, o valor da Selic é o Copom (Comitê de Política Monetária do Banco Central), que a cada 45 dias se reúne para determinar se a taxa aumenta ou diminui. Na prática, se a taxa está alta, os financiados podem ficar mais caros e o contrário também, se a taxa está mais baixa, então os financiamentos ficam mais baratos. Resumindo, quando a taxa Selic aumenta a economia desacelera, quando ela abaixa a economia aquece, isso é preciso para controlar a inflação.

\_\_\_\_\_\_\_

Agora que já conhecemos que a taxa Selic influencia, nossos financiamentos, investimentos e até mesmo o que compramos no mercado, vamos extrair 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\_json() e guardando em um DataFrame (DF) pandas. Ao carregar os dados em um DF, podemos visualizar quantas linhas e colunas, bem como, os tipos de dados em cada coluna, com o método info(). Observe a entrada 4 e a saída do código a seguir, o método retorna que o DF possui 8552 registros (entradas) e 2 colunas (A quantidade de linhas certamente será diferente quando executar esse código). Os índices são numéricos e variam de 0 a 8551 (N-1). Outras duas informações relevantes que esse método retorna é sobre a quantidade de células sem valor (non-null) e o tipo dos dados nas colunas. Como podemos ver, ambas colunas possuem 8552 valores não nulos, como esse número é igual a quantidade de linhas, então não existem valores faltantes. Quanto ao tipo de dados, o "data" é um object, ou seja, são todos do tipo strings ou existe mistura de tipos; "valor" é um float.

In [4]:

df\_selic **=** pd**.**read\_json("https://api.bcb.gov.br/dados/serie/bcdata.sgs.11/dados?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 tratamentos que devemos fazer é remover os dados duplicados. Certamente, qual registro remover depende da área de negócio e do problema a ser resolvido. Por exemplo, 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 dados duplicados.

Observe a entrada 5 da linha de código a seguir, usamos o método para remover as linhas duplicadas, pedindo para manter o último registro (keep='last') e, através do parâmetro inplace=True, estamos fazendo com que a transformação seja salva do DataFrame, na prática estamos sobrescrevendo o objeto na memória. **Caso o inplace não seja passado, a transformação é aplicada, mas não é salva, ou seja, o DF continua da mesma forma anterior a transformação**. Outro parâmetro interessante do método é o subset, que permite que a remoção de dados duplicados seja feita com base em uma ou mais colunas.

In [5]: df\_selic**.**drop\_duplicates(keep**=**'last', inplace**=**True)

**Criar novas colunas e Método TO\_DATETIME() e ASTYPE**



A segunda transformação que veremos é como criar uma nova coluna. A sintaxe é similar a criar uma nova chave em um dicionário: meu\_df['nova\_coluna'] = dado. Vamos criar uma coluna que adiciona a data de extração das informações. Observe a seguir, do módulo datetime, estamos usando a classe date e o método today().

Na entrada 6, guardamos a data em uma nova coluna, veja que a biblioteca pandas "entende" que esse valor deve ser colocado em todas as linhas. Na linha 7, estamos criando uma nova coluna com o nome do responsável pela extração, veja que basta atribuir a string a uma nova coluna. Temos ainda um problema com o tipo de dados das datas, embora cada valor seja do tipo "date", veja que pelo info() ainda obtemos uma coluna object, para que de fato, a biblioteca interprete como um tipo data, vamos ter que utilizar o método da própria biblioteca para fazer 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

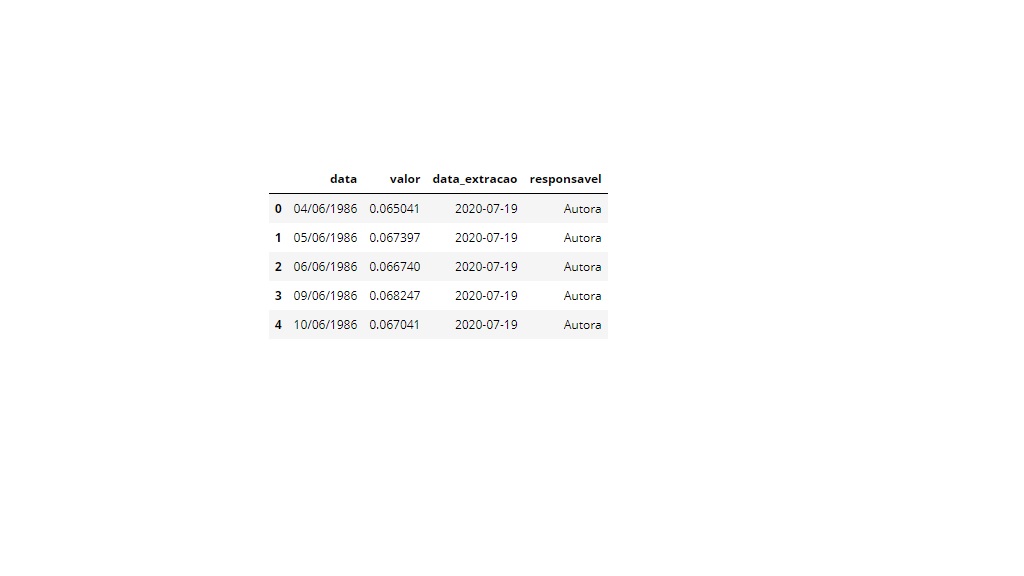
responsavel      8552 non-null object

dtypes: float64(1), object(3)

memory usage: 334.1+ KB

None

Out [6]:



**Método TO\_DATETIME() e ASTYPE** - Trabalhar com o tipo "data" pode trazer vantagens, como por exemplo, ordenar da data mais recente para mais antiga, ou ainda verificar a variação da taxa selic em um determinado período. Vamos utilizar os métodos pandas.to\_datime() e minha\_series.astype() para fazer a conversão e transformar as colunas data e data\_extracao. Observe o código a seguir, onde realizamos a transformação e guardamos dentro da própria coluna, dessa forma os valores são sobrescritos. Na linha 1, usamos a notação pd.to\_datetime(), porque é um método da biblioteca e não do DF.

Na entrada 7 (linha 1), o método recebe a coluna com os valores a serem alterados (df\_selic['data']) e um segundo parâmetro, indicando que no formato atual (antes da conversão) o dia está primeiro (dayfirst=True). Em seguida, na linha 2, como a coluna "data\_extracao" foi criada com o método today(), o formato já está correto para a conversão. Nessa conversão usamos o método astype, que transforma os dados de uma coluna (que é uma Series) em um determinado tipo, nesse caso, o tipo datetime especificado.

Com astype() podemos padronizar valores das colunas, por exemplo, transformando todos em float, ou int, ou str, ou outro tipo. Veja que agora, ao usar o método info(), temos que ambas colunas são do tipo datetime (datetime da biblioteca pandas). O formato resultante ano-mês-dia é um padrão do datetime64[ns], que segue o padrão internacional, no qual o ano vem primeiro, seguido do mês e por último o dia. Poderíamos usar o stftime() para transformar o traço em barra (/), mas aí o resultado seriam strings e não 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())

df\_selic**.**head()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 8552 entries, 0 to 8551

Data columns (total 4 columns):

data             8552 non-null datetime64[ns]

valor            8552 non-null float64

data\_extracao    8552 non-null datetime64[ns]

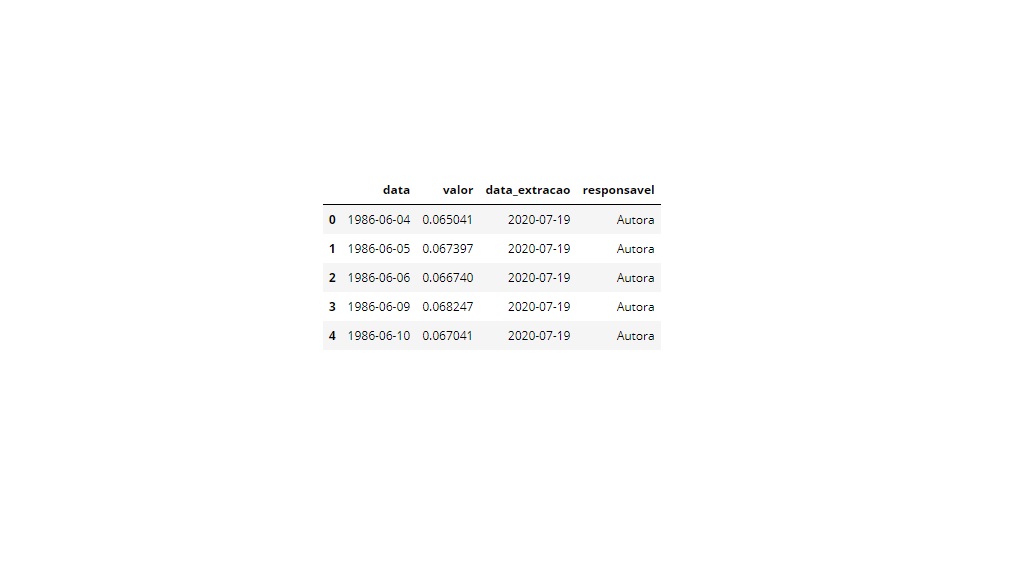
responsavel      8552 non-null object

dtypes: datetime64[ns](2), float64(1), object(1)

memory usage: 334.1+ KB

None

Out [7]:



**Series.str e Método RESET\_INDEX() e SET\_INDEX()**



Muitas vezes precisamos padronizar os valores em colunas, por exemplo, queremos ter certeza que a coluna "responsável" possui todas as palavras padronizadas em letras maiúsculas. Quando selecionamos uma coluna no DF sabemos que o resultado é uma Series e esse objeto tem um recurso "str", que permite aplicar as funções de string para todos os valores da Series.

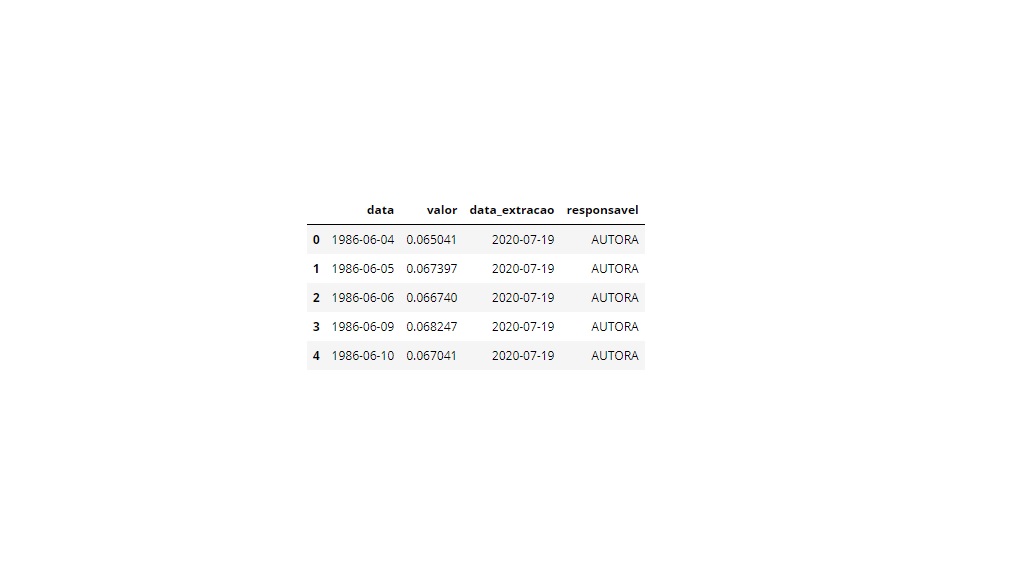
Observe o trecho de código seguir, selecionamos a coluna responsável, acessamos o recurso str e aplicamos o método upper(). Dessa forma, a biblioteca pandas "entende" que queremos converter todos os valores dessa coluna para letras maiúsculas. Como atribuímos o resultado na própria coluna, o valor antigo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

In [8]:

 df\_selic['responsavel'] **=** df\_selic['responsavel']**.**str**.**upper()

df\_selic**.**head()

Out [8]:

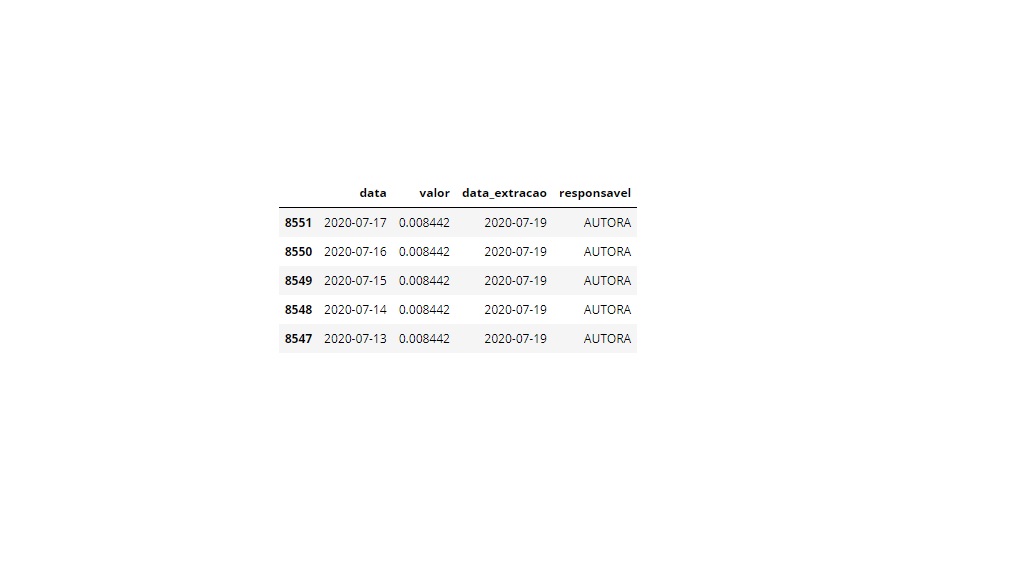


**Método SORT\_VALUES()** - No código a seguir, estamos usando o método sort\_values() que permite ordenar o DF, de acordo com os valores de uma coluna. Esse método é do DataFrame, por isso a notação meu\_df.metodo(). Utilizamos três parâmetros do método sort\_values, o primeiro informando qual coluna deve ser usada para ordenar, o segundo, para que seja feito em ordem decrescente (do maior para o menor) e o terceiro (inplace=True) significa que queremos modificar o próprio objeto, na prática estamos sobrescrevendo o DF.

In [9]: df\_selic**.**sort\_values(by**=**'data', ascending**=**False, inplace**=**True)

df\_selic**.**head()

Out [9]:



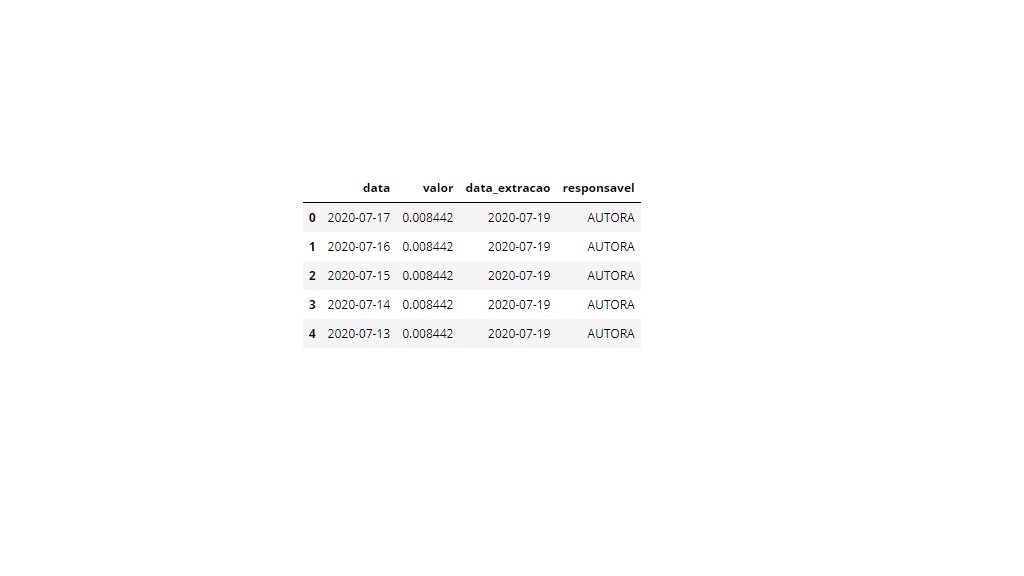
Método RESET\_INDEX() e SET\_INDEX() -Ao fazermos a ordenação dos dados com o método sort\_values(), veja que os índices dos cinco primeiros registros é 8551, 8550...85XX. Nenhuma transformação afeta o índice, lembra-se como não especificamos rótulos ele usa um intervalo numérico, mas esse intervalo**é diferente da posição de um vetor**, pois é um nome e vai acompanhar a linha independente da transformação.

As únicas formas de alterar o índice são com os métodos reset\_index() e set\_index(). O primeiro redefine o índice usando o padrão e o segundo utiliza definições de novos índices. Veja o código na entrada 10, estamos usando o método reset\_index() para redefinir os índices padrão (sequência numérica). O primeiro parâmetro (drop=True), diz que não queremos usar o índice que será substituído em uma nova coluna e inplace, informa para gravar as alterações no próprio objeto. Veja na saída que agora os cinco primeiros registros, possuem índices de 0 a 4.

In [10]: df\_selic**.**reset\_index(drop**=**True, inplace**=**True)

df\_selic**.**head()

Out [10]:



Durante a transformação dos dados, pode ser necessário definir novos valores para os índices, ao invés de usar o range numérico. Essa transformação pode ser feita usando o método meu\_df.set\_index(). O método permite especificar os novos valores usando uma coluna já existente ou então passando uma lista, de tamanho igual a quantidade de linhas.

Observe os códigos nas entradas 11 e 12. Na entrada 11, estamos criando uma lista que usa os índices do DF, adicionando um prefixo para cada valor. Na linha 2 são impressos os cinco primeiros itens da nova lista. Na entrada 12, estamos definindo o novo índice com base na lista criada. Veja que o parâmetros keys, recebe como parâmetro uma lista de lista e o segundo parâmetro especifica que a modificação deve ser salva no objeto. Na impressão das cinco primeiras linhas do DF, podemos ver o novo índice.

In [11]: lista\_novo\_indice **=** [f'selic\_{indice}' for indice in df\_selic**.**index]

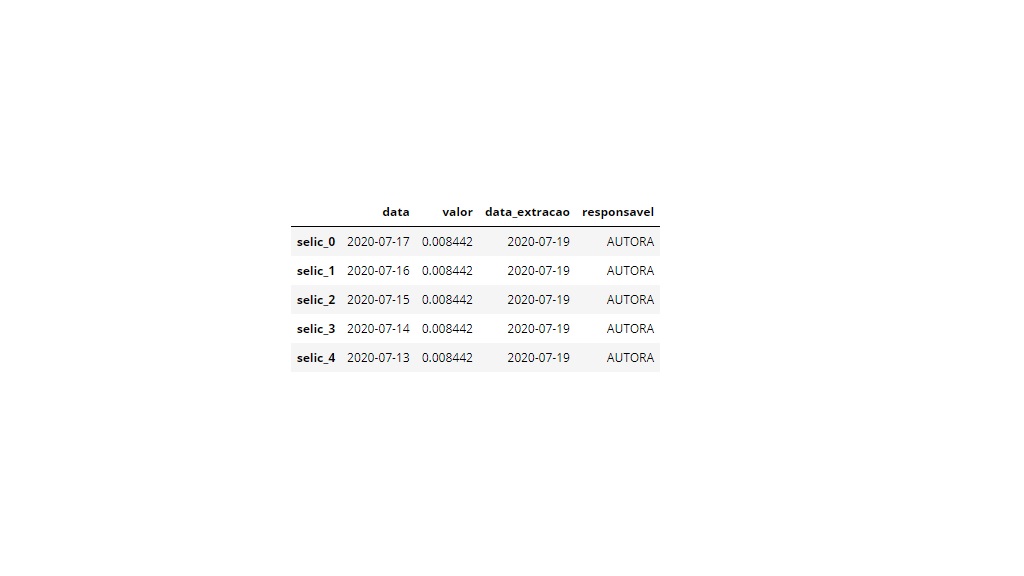
print(lista\_novo\_indice[:5])

['selic\_0', 'selic\_1', 'selic\_2', 'selic\_3', 'selic\_4']

In [12]: df\_selic**.**set\_index(keys**=**[lista\_novo\_indice], inplace**=**True)

df\_selic**.**head()

Out [12]:



Ao especificar um índice com valor conceitual, podemos usar as funções idxmin() e idxmax() para descobrir qual o índice do menor e do maior de uma Series. Observe os 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 passar a fase de extração de informações.

**Filtros com LOC** - Um dos recursos mais utilizados por equipes das áreas de dados é a aplicação de filtros. Imagine a seguinte situação, uma determinada pesquisa quer saber qual é a média de idade de todas as pessoas na sua sala de aula, bem como a média de idades somente das mulheres e somente dos homens. A distinção por gênero é um filtro! Esse filtro vai possibilitar comparar a idade de todos, com a idade de cada grupo e entender se as mulheres ou homens estão abaixo ou acima da média geral.

DataFrames da biblioteca pandas possuem uma propriedade chamada [loc](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html). Essa propriedade permite acessar um conjunto de linhas (filtrar linhas), por meio do índice ou por um vetor booleano (vetor de True ou False).

Vamos começar explorando o filtro pelos índices. Entrada 14 estamos localizando (loc), o registro que possui índice 'selic\_0', como resultado obtém-se uma Series. Na entrada 15, foram filtrados três registros, para isso foi necessário passar uma lista 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

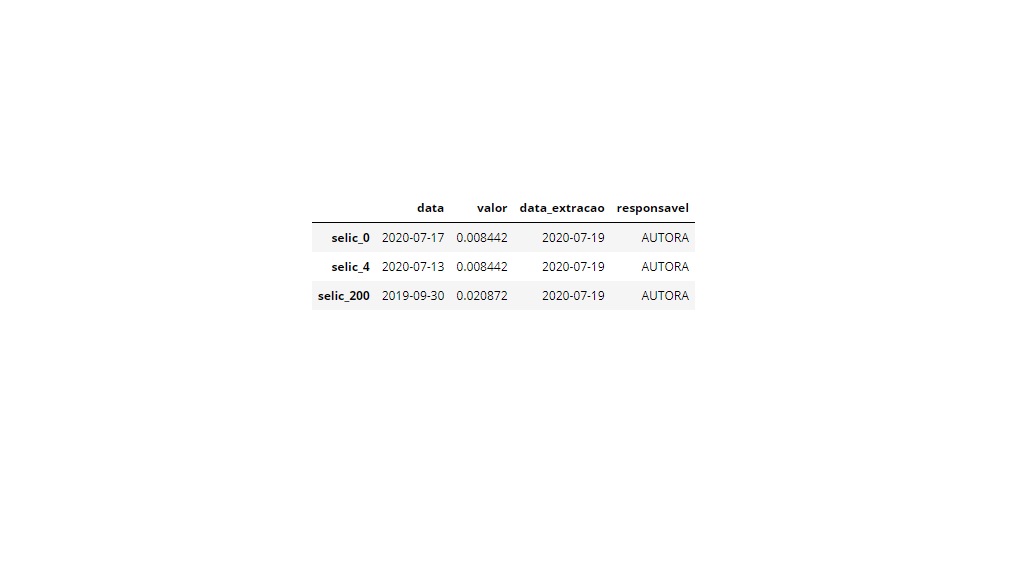
data\_extracao    2020-07-19 00:00:00

responsavel                   AUTORA

Name: selic\_0, dtype: object

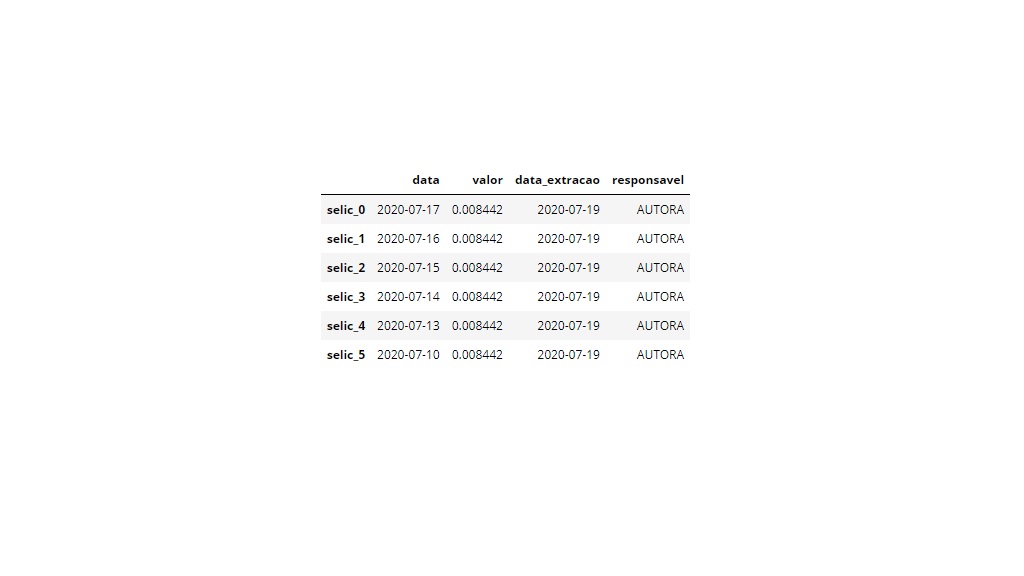
In [15]: df\_selic**.**loc[['selic\_0', 'selic\_4', 'selic\_200']]

Out [15]:



In [16]: df\_selic**.**loc[:'selic\_5']

Out [16]:



A propriedade loc, aceita um segundo argumento, que é a coluna (ou colunas) que serão exibidas para os índices escolhidos. Na entrada 17 selecionamos uma única coluna e na entrada 18 uma lista de colunas.

O mesmo resultado poderia ser alcançado passando a coluna, ou a lista de colunas conforme as sintaxes:

* 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'], 'valor']

In [17]: selic\_0      0.008442

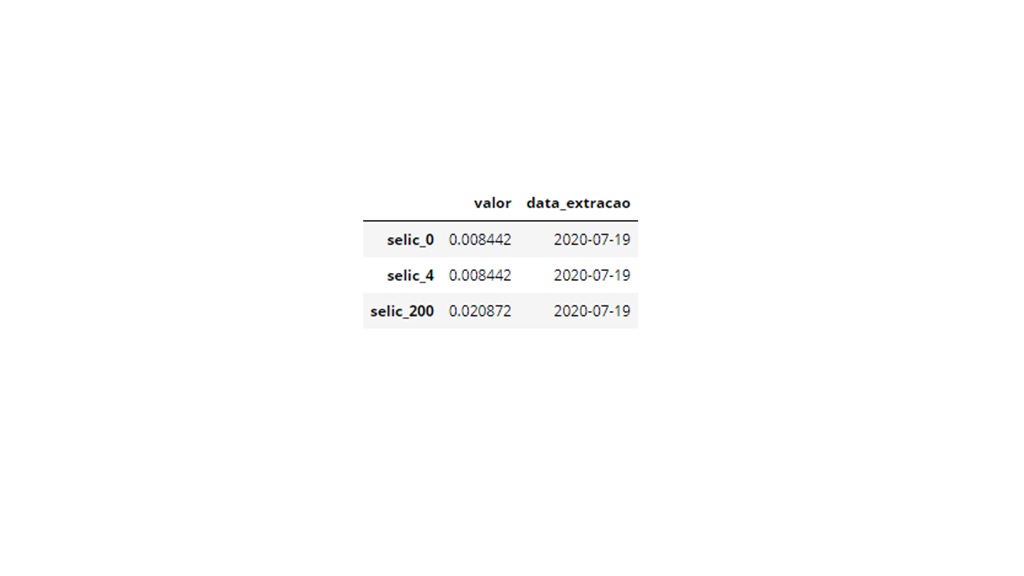
selic\_4      0.008442

selic\_200    0.020872

Name: valor, dtype: float64

In [18]: df\_selic**.**loc[['selic\_0', 'selic\_4', 'selic\_200'], ['valor', 'data\_extracao']]

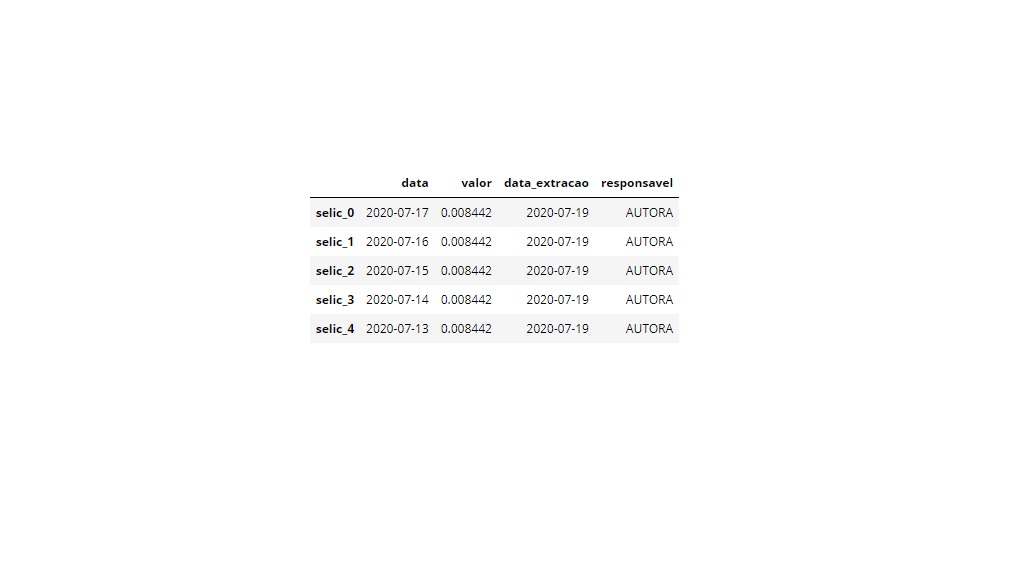
Out [18]:



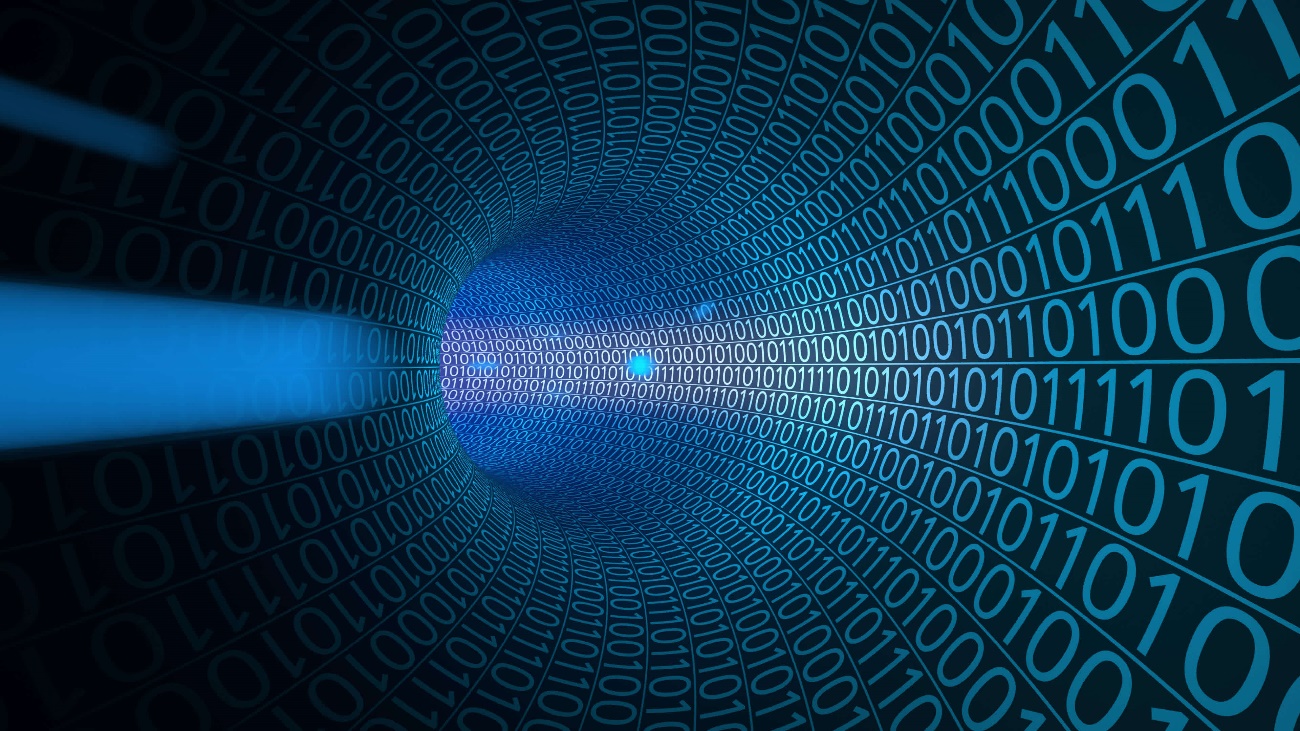
Antes de vermos a criação de filtros para o loc com condições booleanas, vale mencionar que existe também a propriedade iloc, a qual filtra as linhas considerando a posição que ocupam no objeto. Veja no exemplo a seguir, estamos usando o iloc para filtrar os 5 primeiros registros, usando a mesma notação do fatiamento de listas. Essa propriedade não possui a opção de também selecionar colunas. Veja um exemplo a seguir.

In [19]: df\_selic**.**iloc[:5]

Out [19]:



**Filtros com testes Booleanos**



Podemos usar operadores relacionais e lógicos para fazer testes condicionais com os valores das colunas de um DF. Ao criarmos um teste condicional, internamente, a biblioteca testa todas as linhas do DF ou da Series, retornando uma Series booleana, ou seja, composta por valores True ou False.

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

Out [21]: <class 'pandas.core.series.Series'>

Out [21]: selic\_0    True

selic\_1    True

selic\_2    True

selic\_3    True

selic\_4    True

Name: data, dtype: bool

O teste condicional pode ser construído também utilizando operadores lógicos. Para a operação lógica AND (E), em pandas, usa-se o caracter &. Para fazer a operação lógica OR (OU), usa-se o caracter |. Cada teste deve estar entre parênteses, senão ocorre um erro. Observe o código a seguir, na entrada 22 e a 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

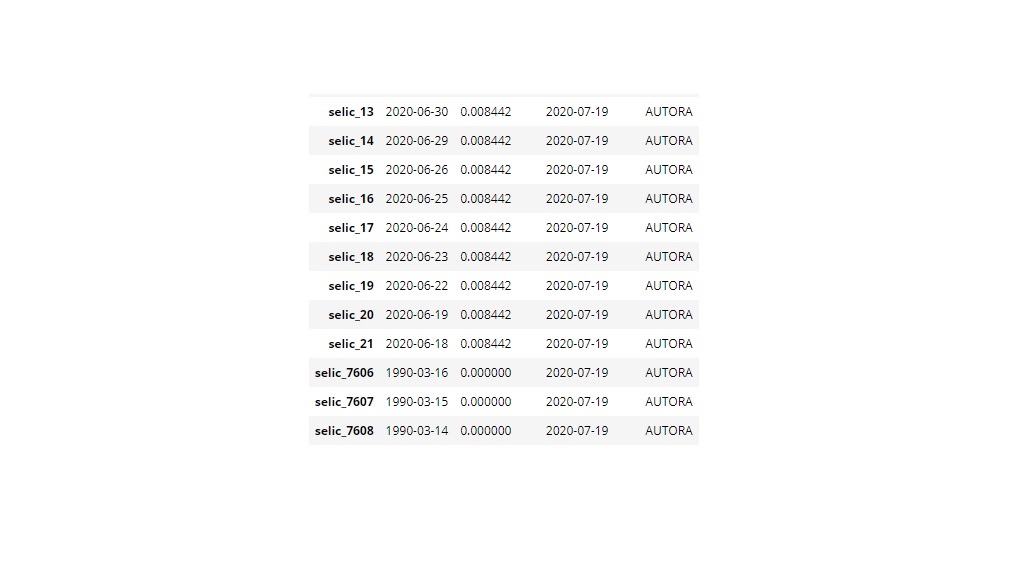
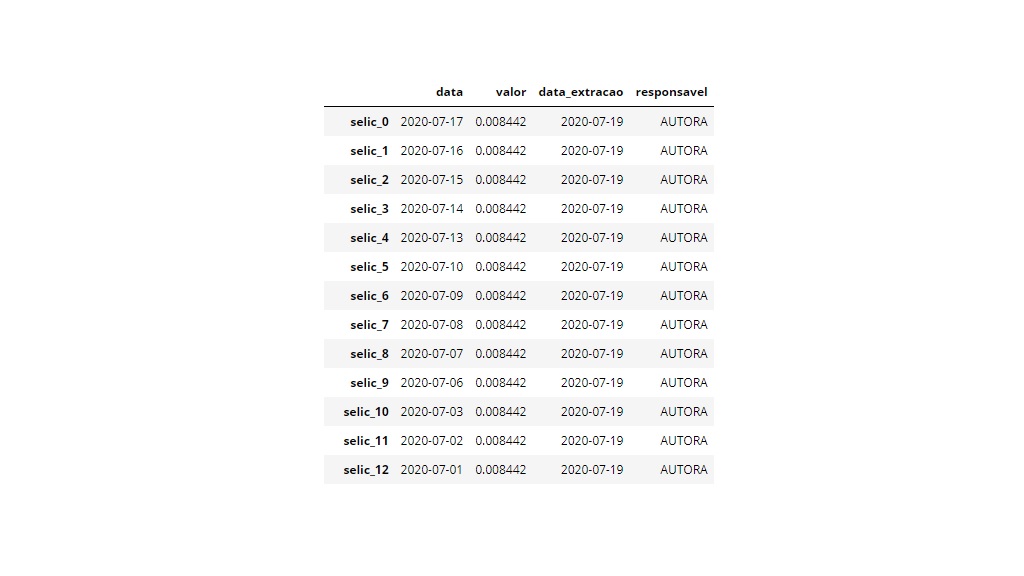
selic\_1    True

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. Observe o código a seguir. Na linha 1 estamos criando a condição do filtro (que é uma Series booleana) e na entrada 23, passamos como parâmetro para filtrar os registros. Esse filtro retornou somente 4 registros.

In [23]:



Na entrada 23, criamos primeiro a condição, guardamos na variável e depois 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 à vontade. Neste conteúdo vamos adotar a sintaxe de criar os filtros e guardar em variáveis por questões didáticas e de legibilidade do código.

Na entrada 24 (linha 1), criamos uma condição para exibir o registro das datas 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 aplicamos no DF, guardando o resultado (que é um DF) em um novo DF. Todo esse filtro poderia ter sido escrito em uma única linha: df\_selic.loc[(df\_selic['data'] >= pd.to\_datetime('2020-01-01')) & (df\_selic['data'] <= pd.to\_datetime('2020-01-31'))], mas veja como fica mais difícil ler e interpretar o filtro, ainda mais para quem for dar manutenção no código.

In [24]: data1 **=** pd**.**to\_datetime('2020-01-01')

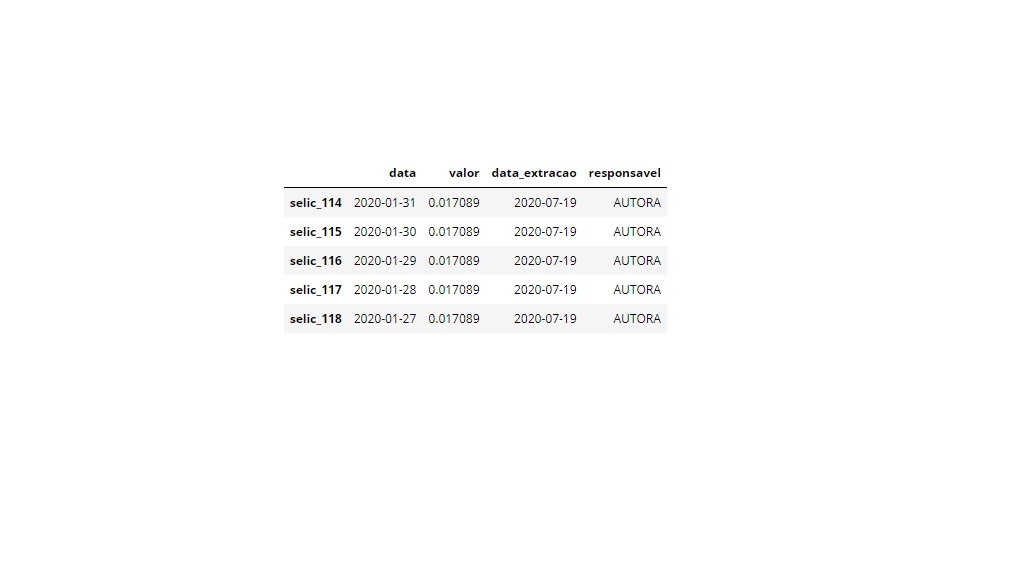
data2 **=** pd**.**to\_datetime('2020-01-31')

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]:



Vamos criar mais um filtro e um novo DF para que possamos ver a importância dos filtros. No código a seguir, vamos criar um novo DF contendo as informações do mês de janeiro do ano de 2019.

In [25]: data1 **=** pd**.**to\_datetime('2019-01-01')

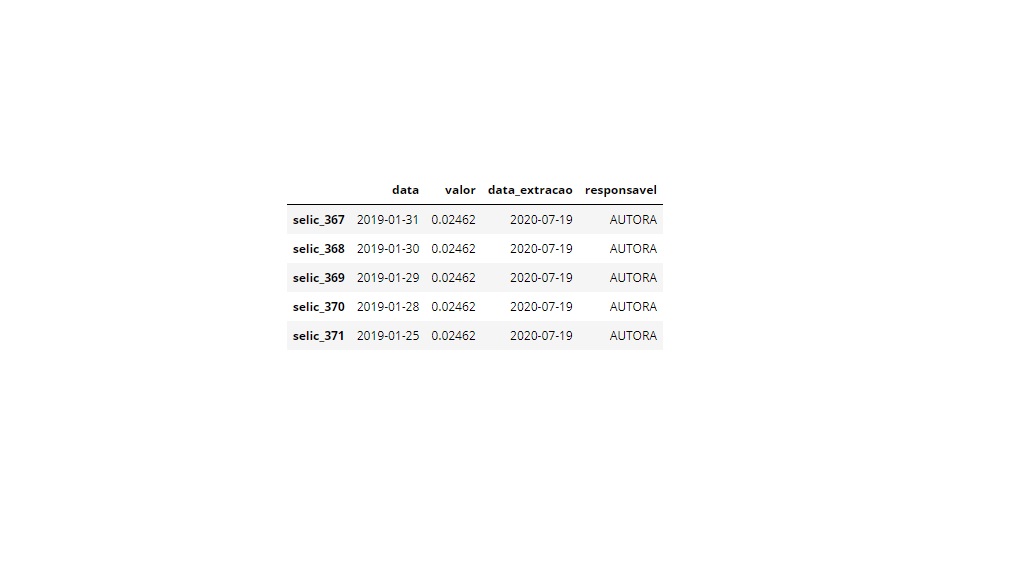
data2 **=** pd**.**to\_datetime('2019-01-31')

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]:



Agora que temos três DFs, um completo e dois com filtros, vamos utilizar métodos da estatística descritiva para extrair informações sobre o valor da taxa selic em cada DF. Queremos saber, qual o valor máximo e mínimo registrado em cada caso e qual a média.

Ao selecionar uma coluna, temos uma Series, então basta utilizar o método solicitado, conforme códigos a seguir.

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(), '\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(), '\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(), '\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 =  0.02461999999999999

Média janeiro de 2020 =  0.017089000000000003

In [27]: df\_selic**.**to\_csv('dados\_selic.csv')

Que tal utilizar o [simulador](https://trinket.io/python3/41267e0e36)a seguir e testar as funcionalidades aprendidas?

Poderíamos extrair diversas outras informações dos dados. Todo esse trabalho faz parte do cotidiano dos engenheiros, cientistas e analistas de dados. Os engenheiros de dados mais focados na preparação e disponibilização dos dados, os cientistas focados em responder questões de negócio, inclusive utilizando modelos de machine learning e deep learning e os analistas, também respondendo a perguntas de negócios e apresentando resultados. Se você gostou do que fizemos quem sabe não deva investir mais nessa área?!

\_\_\_\_\_\_\_

**➕ Pesquise mais**

Existem diversos portais que disponibilizam dados, por exemplo, o próprio kaggle, os portais [brasileiros](http://www.dados.gov.br/dataset%20e%20http:/www.portaldatransparencia.gov.br/), o [portal](https://archive.ics.uci.edu/ml/datasets.php) UCI, ou [o datasets](https://vincentarelbundock.github.io/Rdatasets/datasets.html) vincentarelbundock. Enfim, são inúmeros os repositórios que podem ser explorados, até mesmo o git hub.

**Banco de dados com panda**



Para finalizar nossa aula vamos aprender que além dos métodos para acessar arquivos, a biblioteca pandas possui dois métodos que permitem executar instruções SQL em banco de dados. Os métodos são:

* pandas.read\_sql(sql, con, index\_col=None, coerce\_float=True, params=None, 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 SQL e uma conexão com um banco de dados (con). A conexão com o banco de dados, deve ser feita usando uma outra biblioteca, por exemplo, sqlalchemy (suporte para diversos bancos), pyodbc para SQL Server, cx\_Oracle para Oracle, psycopg2 para Postgresql, dentre outras. Seguindo as recomendações da PEP 249 todas bibliotecas precisam fornecer um método "connect", o qual recebe a string de conexão. A síntaxe da string de conexão depende da biblioteca e do banco de dados.

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

import psycopg2

host = 'XXXXX'

port = 'XXXXX'

database = 'XXXXX'

username = 'XXXXX'

password = 'XXXXX'

conn\_str = fr"dbname='{database}' user='{username}' host='{host}' password='{password}' port='{port}'"

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'

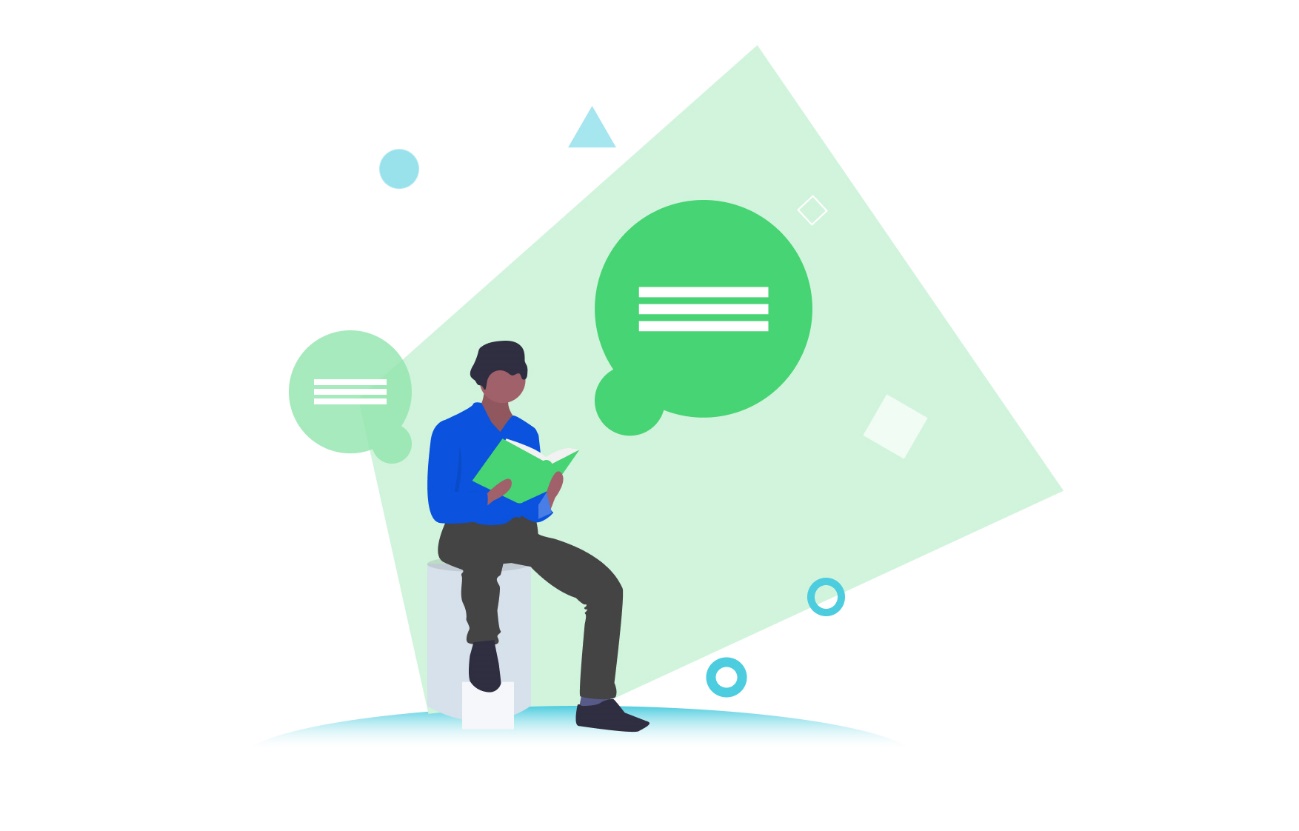
conn\_str = fr"host={host}, user={username}, passwd={password}, database={database}"

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

query = "select \* from XXX.YYYY"

df = pd.read\_sql(query, conn)

# Conclusão



## **Desafio**

Como desenvolvedor em uma empresa de consultoria de software, você foi alocado em um projeto para uma empresa de geração de energia. Essa empresa tem interesse em criar uma solução que acompanhe as exportações de etanol no Brasil. Esse tipo de informação está disponível no site do governo brasileiro <https://dados.gov.br/>, em formatos CSV, JSON, dentre outros.

[Neste link](https://cm-kls-content.s3.amazonaws.com/DESEN_WEBAULA/LDD/20.2/LINGUAGEM_PROGRAMACAO/csv.csv) para download, é possível encontrar uma base de dados (dataset), contendo informações de importação e exportação de produtos de etanol. O cliente está interessado em obter informações sobre a Exportação Etanol Hidratado (barris equivalentes de petróleo) 2012-2023. Para a análise, será necessário fazer o download do arquivo.

O cliente deseja uma solução que extraia as seguintes informações

1. Em cada ano, qual o menor e o maior valor arrecadado da exportação?
2. Considerando o período de 2012 a 2023, qual a média anual de arrecadamento com a exportação?
3. Considerando o período de 2012 a 2023, qual ano teve o maior arrecadamento com a exportação? E o menor?

Como parte das informações técnicas sobre o arquivo, foi-lhe informado que se trata de um arquivo delimitado CSV, cujo separador de campos é ponto-e-vírgula e a codificação do arquivo está em UTF-8. Como podemos obter o arquivo? Como podemos extrair essas informações usando a linguagem Python? Serão necessárias transformações nos dados para obtermos as informações solicitadas?

## **Resolução**

Para começar a resolver o desafio, precisamos fazer o download do arquivo com os dados de importação e exportação de etanol no site <http://www.dados.gov.br/>. Nele, você acessa a seção “Conjunto de dados” (datasets) e pesquisa pela palavra-chave “importação e exportação de etanol”.

início da audiodescrição
    fig1
    fim da audiodescriçãoFigura 1 - Portal Brasileiro de Dados Abertos. Fonte: Captura de Tela do site http://www.dados.gov.br.

Como resultado de sua pesquisa, acesse os dados de Importações e Exportações concedidos pela Agência Nacional do Petróleo, Gás Natural e Biocombustíveis (ANP). Ao clicar no acesso, você terá uma lista de recursos, tanto PDFs com metadados e arquivos .CSV. Ao clicar em “Acessar o recurso” no recurso “Importações e exportações de derivados de etanol (metros cúbicos) 2012-2023”, você fará o download do CSV.

início da audiodescrição
    fig2
    fim da audiodescriçãoFigura 2 - Recurso CSV sobre importações e exportações de etanol. Fonte: Captura de Tela do site http://www.dados.gov.br.

É importante lembrar que este arquivo costuma ser atualizado anualmente. Portanto, para a resolução vigente, utilizaremos o CSV que considera importações e exportações de etanol entre 2012-2023, mantendo os objetivos elencados no Desafio. Após obter o arquivo, basta copiá-lo para a pasta do projeto.

Conforme orientações, o arquivo é delimitado, mas seu separador padrão é o ";" e a codificação do arquivo foi feita em UTF-8. Portanto, teremos que passar esses dois parâmetros para a leitura do arquivo usando a biblioteca pandas, uma vez que o delimitar padrão da biblioteca é o ",". No código a seguir, estamos importando os dados e exibindo as 6 colunas na sua totalidade e, apenas como teste inicial, exibindo as primeiras 24 linhas.

import pandas as pd

# Leia o arquivo CSV com a codificação UTF-8

df\_etanol = pd.read\_csv('importacoes-exportacoes-etanol-2012-2023.csv', sep=';', encoding="utf-8")

# Selecione as importações e exportações das primeiras 24 linhas (corresponde a 1 ano)

df\_etanol\_ano = df\_etanol.head(24)

início da audiodescrição
    fig3
    fim da audiodescrição

Agora que temos os dados, vamos dividir nossa solução em duas etapas: (I) a de transformação dos dados e (II) a de extração de informações.

## I. Etapa de transformações

Vamos começar removendo as colunas e as linhas cujo os dados não serão utilizados, afinal, o cliente está apenas interessado em OPERAÇÕES COMERCIAIS de EXPORTAÇÃO e produto do tipo ETANOL HIDRATADO. Dessa forma, poupamos a memória RAM.

O código a seguir efetua uma filtragem no DF “df\_etanol”, mantendo as linhas que satisfaçam dois critérios. O primeiro critério requer que na coluna “OPERAÇÃO COMERCIAL”, as linhas contenham exclusivamente a string “EXPORTAÇÃO” e o segundo critério requer que na coluna “PRODUTO”, as linhas incluem apenas a string “ETANOL HIDRATADO”. Dessa forma, o DF conterá apenas as informações relevantes para a resolução do problema.

df\_etanol = df\_etanol[~df\_etanol['OPERAÇÃO COMERCIAL'].str.contains('IMPORTAÇÃO')]

df\_etanol = df\_etanol[~df\_etanol['PRODUTO'].str.contains('ETANOL HIDRATADO')]

Depois de remover as linhas, procedemos com a exclusão das colunas “PRODUTO” e “DISPÊNDIO/RECEITA” de “df\_etanol”. O parâmetro “inplace=True” é utilizado para que a transformação seja salva no próprio objeto.

df\_etanol.drop(columns=['PRODUTO', 'DISPÊNDIO / RECEITA'], inplace=True)

Agora, vamos redefinir os índices do Data Frame (DF), com base na coluna ANO. Esse passo será importante para a fase de extração de informações. Veja que também optamos em remover a coluna da tabela pelo parâmetro “drop=True” e novamente usando o parâmetro “inplace=True”.

df\_etanol.set\_index(keys='ANO', drop=True, inplace=True)

Uma vez que os dados são de origem brasileira, a vírgula é usada como separador decimal, o que não condiz com o padrão da biblioteca pandas. Para converter todas as vírgulas em pontos na coluna “IMPORTADO/EXPORTADO”, utilizamos a funcionalidade “str.replace(‘,’,’.’)”.

Mesmo após essa substituição, a biblioteca pandas ainda interpreta os valores como objetos em vez de números float. Para corrigir essa interpretação, aplica-se o método “astype(float)”, que converte os valores do df\_etanol para o formato desejado.

df\_etanol['IMPORTADO / EXPORTADO'] = df\_etanol['IMPORTADO / EXPORTADO'].str.replace(',', '.', regex=True).astype(float)

Veja que trabalhamos muito com tabelas usando data frames. Para apresentar os resultados de forma mais organizada e legível, poderíamos utilizar a biblioteca tabulate. Que tal se aprofundar e pesquisar mais?

## **II. Etapa de extração de informações**

Agora que preparamos os dados, podemos começar a etapa de extração das informações solicitadas, respondendo às solicitações do cliente.

### **A. Em cada ano, qual o menor e o maior valor arrecadado da exportação?**

Como nosso índice é o próprio ano, podemos usar a função “loc” para filtrar todas as linhas relacionadas ao ano e, em seguida, os métodos min() e max() para encontrar os valores mínimo e máximo na coluna “IMPORTADO/EXPORTADO”. Para realizar essa extração para todos os anos, usamos uma estrutura de repetição.

Nas impressões ‘f"{maximo:,.0f}".replace(',', '.')’ e ‘f"{minimo:,.0f}".replace(',', '.')’, estamos imprimindo os valores solicitados. A expressão ‘{maximo:,.0f}’ faz com que apenas a parte inteira dos números seja exibida, utilizando vírgulas como separadores de milhares. Em seguida, substituímos a vírgula por um ponto, que é o padrão brasileiro, para a apresentação final dos valores.

início da audiodescrição
    codigo 1
    fim da audiodescrição

Saída:

Ano 2012

Maior valor -> DEZ = 213.683

Menor valor -> JAN = 23.240

--------------

Ano 2013

Maior valor -> JAN = 179.411

Menor valor -> ABR = 14.490

--------------

Ano 2014

Maior valor -> JAN = 92.986

Menor valor -> MAI = 19.796

--------------

Ano 2015

Maior valor -> DEZ = 171.143

Menor valor -> ABR = 8.430

--------------

Ano 2016

Maior valor -> FEV = 192.061

Menor valor -> DEZ = 20.054

--------------

Ano 2017

Maior valor -> AGO = 85.619

Menor valor -> FEV = 710

--------------

Ano 2018

Maior valor -> OUT = 127.495

Menor valor -> FEV = 1.132

--------------

Ano 2019

Maior valor -> AGO = 89.933

Menor valor -> ABR = 3.970

--------------

Ano 2020

Maior valor -> DEZ = 220.666

Menor valor -> MAR = 22.336

--------------

Ano 2021

Maior valor -> JUN = 189.334

Menor valor -> MAI = 16.955

--------------

Ano 2022

Maior valor -> DEZ = 162.472

Menor valor -> MAI = 12.134

--------------

Ano 2023

Maior valor -> AGO = 179.405

Menor valor -> JUN = 12.765

--------------

### B. Considerando o período de 2012 a 2023, qual a média anual de arrecadamento com a exportação?

Novamente, usamos o método “loc” para filtrar os dados por ano e, em seguida, calcular a média dos valores utilizando o método “mean()”. Isso nos permite obter a média anual das quantidades de “IMPORTADO/EXPORTADO” para cada ano no intervalo de 2012 a 2023.

início da audiodescrição
    codigo 2
    fim da audiodescrição

Os resultados são impressos, exibindo o ano correspondente e a média anual:

Ano = 2012

Média Anual = 90.541

----------------

Ano = 2013

Média Anual = 92.548

----------------

Ano = 2014

Média Anual = 53.419

----------------

Ano = 2015

Média Anual = 69.716

----------------

Ano = 2016

Média Anual = 75.360

----------------

Ano = 2017

Média Anual = 29.693

----------------

Ano = 2018

Média Anual = 51.285

----------------

Ano = 2019

Média Anual = 51.436

----------------

Ano = 2020

Média Anual = 121.556

----------------

Ano = 2021

Média Anual = 111.206

----------------

Ano = 2022

Média Anual = 90.888

----------------

Ano = 2023

Média Anual = 105.581

----------------

### C. Considerando o período de 2012 a 2023, qual ano teve o maior arrecadamento com a exportação? E o menor?

Para isso, novamente utilizamos um loop para percorrer cada ano e a função “loc”. Entretanto, desta vez, aplicamos o método “sum()” para calcular a soma total. Após essa etapa, é verificado se o valor encontrado é o ano com o maior ou o menor arrecadamento.

início da audiodescrição
    codigo 3
    fim da audiodescrição

Após iterar por todos os anos, o resultado é impresso:

Ano com o menor valor somado: 2017 - Valor: 356.315

Ano com o maior valor somado: 2020 - Valor: 1.458.672

**Dica!** Como a tabela possui mais de 500 linhas e seria custoso verificar se os valores estão corretos, você pode abrir o CSV completo em um editor de planilhas, como o Excel, e utilizar as funções conforme cada solicitação.

Veja que as 3 solicitações do cliente foram divididas em funções, aprimorando a organização do código. Agora, estudante, é a sua vez. Será possível otimizar o código e reescrever algumas partes usando métodos diferentes do pandas e do DataFrame para melhorar o desempenho e a legibilidade? Ou reescrever as funções visando maior eficiência?