## **Capítulo 58. Manipulação de dados, tratamento de dados**

Luiz Paulo O. Paula, Bruno Hanai, Davi Franzão e Joan Davi

Exemplos adicionais do capítulo.

### NumPy

#### Criação de Arrays Bidimensionais

O exemplo a seguir ilustra a criação de um array bidimensional:

|  |
| --- |
| # Criando array bidimensional array2 = np.array([[10, 23, 15],[22, 48, 1]])  array2 |

Saída:

|  |
| --- |
| array([[10, 23, 15],  [22, 48, 1]]) |

O atributo shape do array2 exibe a definição de uma matriz com 2 linhas e 3 colunas:

|  |
| --- |
| # Verificando o formato (shape) do array array2.shape |

Saída:

|  |
| --- |
| (2, 3) |

#### Operações com Matrizes

##### Subtração de matrizes

|  |
| --- |
| A = np.array([[1,0],[0,1]]) B = np.array([[1,2],[3,4]])  np.subtract(A,B) |

Saída:

|  |
| --- |
| array([[ 0, -2],  [-3, -3]]) |

##### 

##### Multiplicação de matrizes

Multiplicação elemento a elemento:

|  |
| --- |
| A \* B |

Saída:

|  |
| --- |
| array([[1, 0],  [0, 4]]) |

Multiplicação de matrizes:

|  |
| --- |
| A @ B |

Saída:

|  |
| --- |
| array([[1, 2],  [3, 4]]) |

Multiplicação de matrizes (outra forma):

|  |
| --- |
| A.dot(B) |

Saída:

|  |
| --- |
| array([[1, 2],  [3, 4]]) |

##### 

##### Divisão de matrizes

|  |
| --- |
| np.divide(A,B) |

Saída:

|  |
| --- |
| array([[1. , 0. ],  [0. , 0.25]]) |

##### 

##### Matriz Transposta

|  |
| --- |
| B = np.array([[1,2,3]]) B |

Saída:

|  |
| --- |
| array([[1, 2, 3]]) |

Exibindo a transposta da matriz B:

|  |
| --- |
| B.T |

Saída:

|  |
| --- |
| array([[1],  [2],  [3]]) |

##### 

##### Matriz Inversa

|  |
| --- |
| A = np.array([[1,2],[3,4]]) A |

Saída:

|  |
| --- |
| array([[1, 2],  [3, 4]]) |

Exibindo a Inversa da Matriz A:

|  |
| --- |
| np.linalg.inv(A) |

Saída:

|  |
| --- |
| array([[-2. , 1. ],  [ 1.5, -0.5]]) |

#### 

#### Operações com dataframes

**[REVISAR A PARTIR DAQUI]**

#### Interrows e Intertuples

Interrows e Intertuples são métodos para iterar sob as linhas de um dataframe, ambas com o mesmo objetivo mas funcionamento diferentes. Quando chamamos o método .iterrows em um dataframe ele irá nos retornar um *generator*  por meio da palavra chave *yeld.* Vamos a um exemplo nas duas primeiras linha de nosso dataset.

|  |
| --- |
| youtube\_raw.head(2).iterrows() |

|  |
| --- |
| <generator object DataFrame.iterrows at 0x7f2363e369e8> |

Temos nosso generator, agora vamos consumi-lo.

|  |
| --- |
| **for** i **in** youtube\_raw.head(2).iterrows():  print(i[1][1]) |

|  |
| --- |
| (0, Data de trending 17.14.11 Título WE WANT TO TALK ABOUT OUR MARRIAGE Canal CaseyNeistat Data e hora de publicação 2017-11-13T17:13:01.000Z Tags SHANtell martin Visualizações 748374 Gostei 57527 Não gostei 2966 Quantidade de comentários 15954 Descrição SHANTELL'S CHANNEL - https://www.youtube.com/s... Comentário desabilitado 0 Avaliação desabilitada 0 Vídeo com erro ou removido 0 Name: 0, dtype: object) (1, Data de trending 17.14.11 Título The Trump Presidency: Last Week Tonight with J... Canal LastWeekTonight Data e hora de publicação 2017-11-13T07:30:00.000Z Tags last week tonight trump presidency"|"last week... Visualizações 2418783 Gostei 97185 Não gostei 6146 Quantidade de comentários 12703 Descrição One year after the presidential election, John... Comentário desabilitado 0 Avaliação desabilitada 0 Vídeo com erro ou removido 0 Name: 1, dtype: object) |

A cada iteração o *generator* do método .iterrows irá nos retornar um par (indice, *series*) onde essa *series* é o conteúdo de linha que está sendo iterada. Já o método .itertuple irá iterar sob o dataframe por meio de *namedtuples* e nos retorna um iterator. Para exemplificar vamos criar um subset em nosso dataset com as três primeiras linhas das colunas “Título”, “Canal”, “Visualizações” e “Gostei”.

|  |
| --- |
| youtube\_subset = youtube\_raw.iloc[0:3, [1,2,5,6]] youtube\_subset |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Título | Canal | Visualizações | Gostei |
| 0 | WE WANT TO TALK ABOUT OUR MARRIAGE | CaseyNeistat | 748374 | 57527 |
| 1 | The Trump Presidency: Last Week Tonight **with** J... | LastWeekTonight | 2418783 | 97185 |
| 2 | Racist Superman | Rudy Mancuso, King Bach & Le... | Rudy Mancuso | 3191434 | 146033 |

Agora vamos iterar sob nosso novo dataframe.

|  |
| --- |
| **for** i **in** youtube\_subset.itertuples():  print(i) |

|  |
| --- |
| Pandas(Index=0, Título='WE WANT TO TALK ABOUT OUR MARRIAGE', Canal='CaseyNeistat', Visualizações=748374, Gostei=57527) Pandas(Index=1, Título='The Trump Presidency: Last Week Tonight with John Oliver (HBO)', Canal='LastWeekTonight', Visualizações=2418783, Gostei=97185) Pandas(Index=2, Título='Racist Superman | Rudy Mancuso, King Bach & Lele Pons', Canal='Rudy Mancuso', Visualizações=3191434, Gostei=146033) |

Como resultado temos 3 *namedtuples*, uma para cada linha

#### Filtros **[ESTA PARTE INICIAL DEVERIA ESTAR ANTES DA SEÇÃO ANTERIOR - E FALTA PEGAR COM OS AUTORES O ARQUIVO!]**

Para os exemplos a seguir, vamos utilizar os dados que estão no arquivo youtube.csv, disponibilizado no link a seguir: [link]

Esse dataset possui informações (visualizações, likes, dislikes, comentários) de vídeos de alguns canais famosos do YouTube. Essas informações foram observadas diariamente no período entre 14/11/2017 até 14/06/2018.

Primeiro, faça o download do arquivo youtube.csv. Depois, carregue-o utilizando o Pandas:

|  |
| --- |
| # se necessário, ajuste o diretório onde o arquivo foi salvo: youtube\_raw = pd.read\_csv('./youtube.csv', sep=";") |

Os campos "data\_da\_observacao" e "data\_da\_publicacao\_do\_video" contêm valores de data, mas foram lidos como texto.

Isso pode ser observado abaixo:

|  |
| --- |
| youtube\_raw.dtypes |

|  |
| --- |
| data\_da\_observacao object video object canal object data\_da\_publicacao\_do\_video object visualizacoes float64 gostei float64 nao\_gostei int64 quantidade\_de\_comentarios int64 descricao\_do\_video object dtype: object |

Vamos converter esses valores para data e salvar as alterações na variável youtube:

|  |
| --- |
| youtube = youtube\_raw.copy() youtube['data\_da\_observacao'] = pd.to\_datetime(youtube['data\_da\_observacao'], format='%y.%d.%m') youtube['data\_da\_publicacao\_do\_video'] = pd.to\_datetime(youtube['data\_da\_publicacao\_do\_video'], format='%d/%m/%Y') |

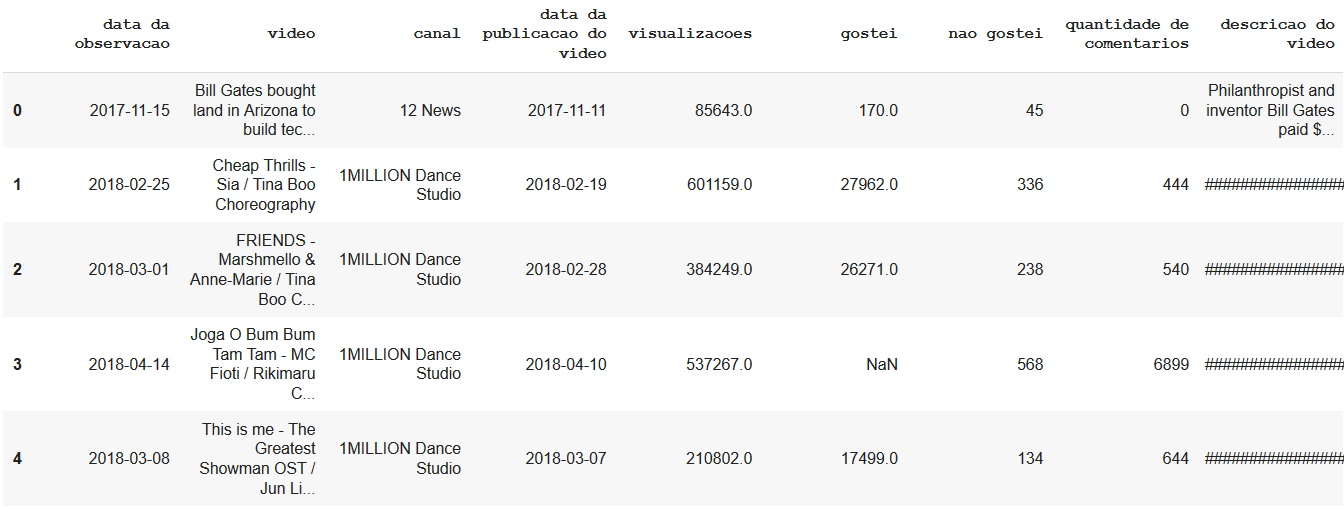
Podemos novamente verificar os tipos dos campos e confirmar que foram convertidos para data:

|  |
| --- |
| youtube.dtypes |

|  |
| --- |
| data\_da\_observacao datetime64[ns] video object canal object data\_da\_publicacao\_do\_video datetime64[ns] visualizacoes float64 gostei float64 nao\_gostei int64 quantidade\_de\_comentarios int64 descricao\_do\_video object dtype: object |

Vamos agora visualizar as primeiras linhas do dataset antes começar a aprender e aplicar filtros:

|  |
| --- |
| youtube.head() |



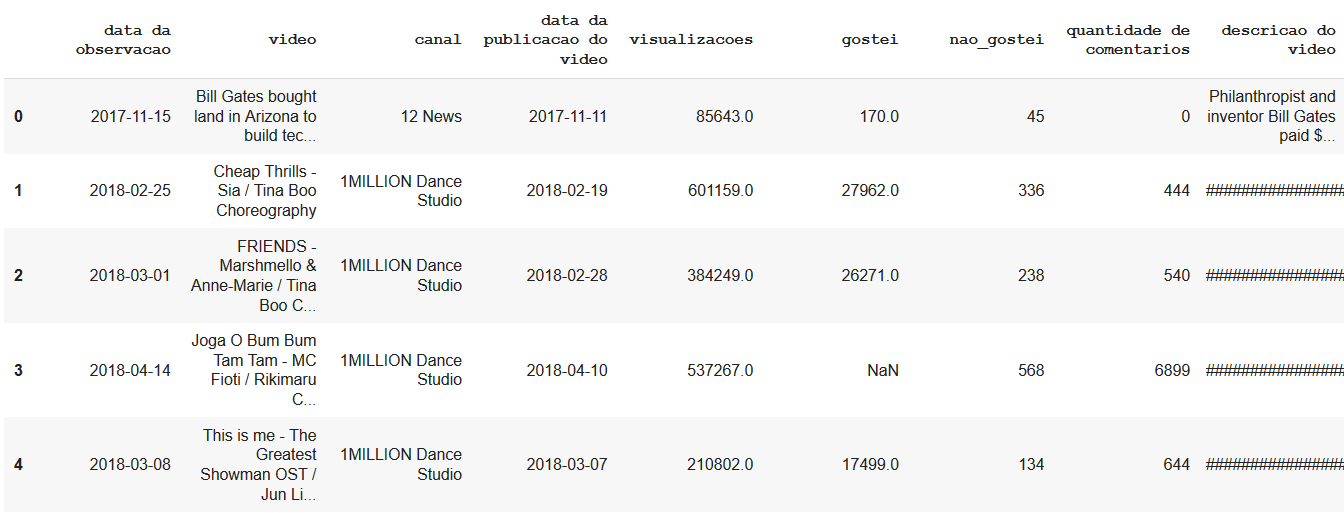
##### Filtros pelo conteúdo das colunas

Nós vamos começar pelo que provavelmente é a forma mais comum de filtro: quando queremos obter somente as linhas de acordo com o conteúdo de uma coluna.

Antes de tudo, precisamos entender como um filtro é feito no Pandas. Para isso, vamos começar com um exemplo simples.

Primeiro, pegue um pequeno pedaço do Dataframe (vamos chamá-lo de youtube\_subset):

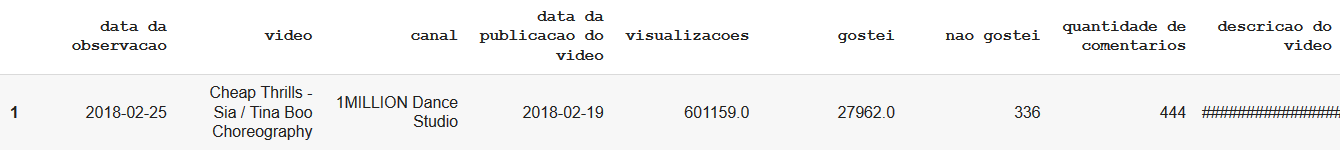
|  |
| --- |
| youtube\_subset = youtube[:5] youtube\_subset |



Então vamos dizer que, por algum motivo qualquer, você está interessado apenas na segunda linha do youtube\_subset. Olhe para o Dataframe na imagem [x], que foi exibido logo acima, e identifique essa linha.

Existem algumas maneiras de obter somente essa segunda linha. Eu vou fazer da forma mais parecida com o filtro:

|  |
| --- |
| manual\_filter = [**False**, **True**, **False**, **False**, **False**] youtube\_subset[manual\_filter] |



Explicando o que fizemos:

1. Criamos uma lista com 5 valores booleanos. A lista tem 5 valores porque essa é a quantidade de linhas do youtube\_subset. Perceba que somente o segundo valor é True, os outros são False.
2. Aplicamos essa lista no youtube\_subset. Somente a linha que fez match com o valor True retornou (a segunda linha).

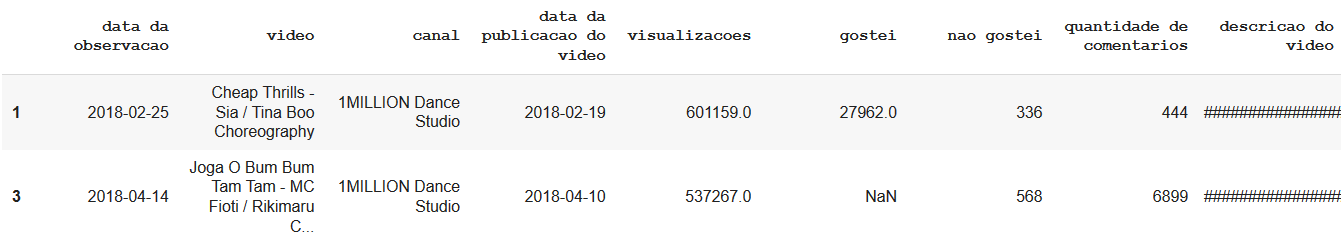
Veja esse match de uma forma mais visual:



Para complementar, vamos ver mais um exemplo. Dessa vez vamos selecionar, além da segunda linha, a quarta linha. Novamente, verifique o Dataframe da imagem [x] para identificar essas linhas.

Segundo filtro manual:

|  |
| --- |
| manual\_filter = [**False**, **True**, **False**, **True**, **False**] youtube\_subset[manual\_filter] |



Novamente, fizemos o match e recebemos apenas os matches com True:



Então, com esses simples exemplos, você deve ter entendido o conceito básico dos filtros no Pandas.

O que você precisa aprender agora é como fazer isso em Dataframes maiores, ou seja, aquelas que contêm muito mais do que apenas 5 linhas. Para fazer isso, a lógica é a mesma: criamos uma lista (pode ser um simples array ou um objeto Series do Pandas) com valores booleanos (True / False), fazemos o match com o Dataframe e então recebemos apenas as linhas onde o match foi feito com um valor True.

O detalhe é que em Dataframes maiores não criamos a lista com booleanos manualmente, pois isso não seria eficiente e em muitas das vezes seria inviável. O que fazemos é aplicar uma condição a uma Series do Pandas (valores de uma coluna de um Dataframe), obtendo assim uma nova Series com valores booleanos pronta para ser usada para filtrar valores.

Para ver como isso funciona na prática, vamos filtrar todos os vídeos do canal "LastWeekTonight".

O nome do canal está na coluna "canal". Usando o comando abaixo, nós recebemos uma Series com os valores dessa coluna:

|  |
| --- |
| youtube['canal'] |

|  |
| --- |
| 0 12 News 1 1MILLION Dance Studio 2 1MILLION Dance Studio 3 1MILLION Dance Studio 4 1MILLION Dance Studio  ...  6435 영국남자 Korean Englishman 6436 圧倒的不審者の極み! 6437 杰威爾音樂 JVR Music 6438 杰威爾音樂 JVR Music 6439 郭韋辰 Name: canal, Length: 6440, dtype: object |

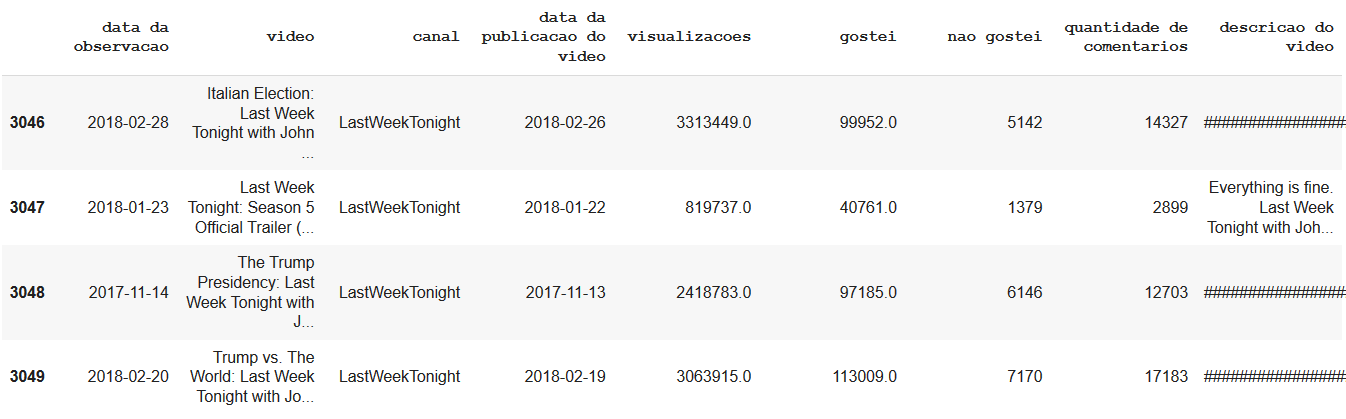
Então podemos usar o operador de comparação == nesse objeto para obter uma nova Series com valores True sempre que a comparação for verdadeira:

|  |
| --- |
| youtube['canal'] == 'LastWeekTonight' |

|  |
| --- |
| 0 **False** 1 **False** 2 **False** 3 **False** 4 **False**  ...  6435 **False** 6436 **False** 6437 **False** 6438 **False** 6439 **False** Name: canal, Length: 6440, dtype: bool |

Nesse momento, talvez você já tenha percebido que isso é o suficiente para fazer o filtro que queremos, ou seja, selecionar apenas os vídeos do canal "LastWeekTonight":

|  |
| --- |
| lwt\_filter = youtube['canal'] == 'LastWeekTonight' youtube[lwt\_filter] |



O que fizemos:

1. Fizemos uma comparação utilizando == e o valor "LastWeekTonight" na coluna "canal". Essa comparação retorna uma Series, que foi armazenada em lwt\_filter.
2. Aplicamos essa Series no Dataframe. Somente as linhas que fizeram match com o valor True retornaram.

Esse é o conceito geral dos filtros baseados em valores de colunas do Pandas. O que vai mudar é a forma que geramos a lista com os valores booleanos. Pode ser de forma manual (como fizemos no 1º exemplo e é uma forma incomum), pode ser com operadores de comparação (como fizemos agora e é a forma mais comum) ou com algumas funções úteis que já vamos ver em breve.

Agora que passamos pelo conceito básico, vamos ver alguns exemplos práticos de filtros. Vou focar em exemplos que você certamente vai precisar no dia a dia.

**Filtros simples, com única condição, utilizando textos ou números**

Abaixo estão alguns exemplos simples de filtros. Alterando os operadores de comparação, as colunas utilizadas e os valores, é possível realizar muitos outros filtros.

Registros onde o nome do vídeo é "clickbait":

|  |
| --- |
| filters = youtube['video'] == 'clickbait' youtube[filters] |

Registros onde o nome do vídeo é diferente de "clickbait":

|  |
| --- |
| filters = youtube['video'] != 'clickbait' youtube[filters] |

Registros com apenas 1 comentário:

|  |
| --- |
| filters = youtube['quantidade\_de\_comentarios'] == 1 youtube[filters] |

Registros com menos de 10 comentários:

|  |
| --- |
| filters = youtube['quantidade\_de\_comentarios'] < 10 youtube[filters] |

Registros com 1.000.000 ou mais comentários:

|  |
| --- |
| filters = youtube['quantidade\_de\_comentarios'] >= 1000000 youtube[filters] |

Registros com o campo "descricao\_do\_video" não informado (exemplo um pouco diferente):

|  |
| --- |
| filters = youtube['Descrição'].isnull() youtube[filters] |

**Filtros simples, com única condição, utilizando datas**

Vamos usar uma lógica parecida com os exemplos anteriores. A única diferença é que o valor usado para fazer a comparação precisa ser um objeto data. Nós podemos utilizar a função to\_datetime do Pandas para criar esse objeto.

Registros com observações feitas em 04/12/2017:

|  |
| --- |
| filters = youtube['data\_da\_observacao'] == pd.to\_datetime('2017-12-04') youtube[filters] |

Registros com observações a partir de 15/02/2018 (inclusive):

|  |
| --- |
| filters = youtube['data\_da\_observacao'] >= pd.to\_datetime('2018-02-15') youtube[filters] |

Registros com observações antes de 15/02/2018:

|  |
| --- |
| filters = youtube['data\_da\_observacao'] < pd.to\_datetime('2018-02-15') youtube[filters] |

##### **Filtros utilizando acessadores de colunas do tipo data**

Quando a coluna contém valores do tipo data, podemos utilizar o acessador .dt, que contém auxiliares muito úteis para pegar partes de uma data.

Registros de vídeos publicados em 2016:

|  |
| --- |
| filters = youtube['data\_da\_publicacao\_do\_video'].dt.year == 2016 youtube[filters] |

Registros de vídeos publicados no mês 11 (ignorando o ano):

|  |
| --- |
| filters = youtube['data\_da\_publicacao\_do\_video'].dt.month == 11 youtube[filters] |

Registros de vídeos publicados no dia 1 (ignorando o mês e ano):

|  |
| --- |
| filters = youtube['data\_da\_publicacao\_do\_video'].dt.day == 1 youtube[filters] |

##### **Filtros utilizando acessadores de colunas do tipo texto**

Para colunas com valores do tipo string, temos o acessador .str, que fornece diversas funções úteis para tratar os textos e criar condições de filtros.

Uma das funções mais úteis para filtros é a .contains. Abaixo, deixo alguns exemplos úteis.

Registros de vídeos que contêm "$" no nome (usamos regex=**False** porque realmente queremos o caractere "$" e não o que ele significa em expressão regular):

|  |
| --- |
| filters = youtube['video'].str.contains('$', regex=**False**) youtube[filters] |

Registros de vídeos que terminam com "." (aqui usamos expressão regular):

|  |
| --- |
| filters = youtube['video'].str.contains('[^.][.]{1}$', regex=**True**) youtube[filters] |

Além do .contains, vários outras funções estão disponíveis no acessador .str. Vamos ver alguns deles nos exemplos abaixo.

Registros de vídeos que o título tem 10 caracteres:

|  |
| --- |
| filters = youtube['video'].str.len() == 10 youtube[filters] |

Registros de vídeos que o título é formado somente por números:

|  |
| --- |
| filters = youtube['video'].str.isdigit() youtube[filters] |

Registros de vídeos que o título é formado por letras maiúsculas:

|  |
| --- |
| filters = youtube['Título'].str.isupper() youtube[filters] |

**Filtros com múltiplas condições**

Os exemplos que vimos até agora têm algo em comum: são formados por uma única condição.

Muitas vezes precisamos realizar filtros múltiplos. São filtros com mais de 1 condição, utilizando valores de mais de 1 coluna, ou ambos os casos.

Vamos pensar em alguns casos utilizando apenas 1 coluna:

* O nome do canal é "LastWeekTonight", "TED" ou "Google"
* O vídeo tem 1.000, 2.000 ou 3.000 comentários
* O vídeo foi publicado em 01/12/2017, 05/12/2017 ou 10/12/2017

E agora alguns casos utilizando mais de 1 coluna:

* O nome do canal é LastWeekTonight e a quantidade de comentários é maior que 100
* A quantidade de visualizações é maior que 1.000 e menor que 10.000
* O nome do canal é LastWeekTonight e a quantidade de comentários é maior que 100
* Todas as letras são maiúsculas **ou** todas as letras são minúsculas

Agora que você já sabe realizar filtros simples, não será difícil fazer os filtros compostos. Você só precisa combinar os filtros utilizando os operadores booleanos: & para expressar "and", | para expressar "or".

Antes de tudo, quero informá-lo do seguinte: se as condições são aplicadas em apenas 1 coluna, muitas vezes não precisamos dos operadores booleanos.

Isso porque esses casos podem ser resolvidos utilizando a função .isin. Para mostrar como isso funcionar, preparei os exemplos abaixo.

Registros dos canais 3Blue1Brown, TED e Google:

|  |
| --- |
| filters = youtube['canal'].isin(['3Blue1Brown', 'TED', 'Google']) youtube[filters] |

Registros de vídeos com 1.000, 2.000 ou 3.000 comentários:

|  |
| --- |
| filters = youtube['Quantidade de comentários'].isin([1000, 2000, 3000]) youtube[filters] |

Registros de vídeos publicados nos dias 01/12/2018, 05/12/2018 ou 10/12/2018:

|  |
| --- |
| filters = youtube['data\_da\_publicacao\_do\_video'].isin([  pd.to\_datetime('2018-12-01'),   pd.to\_datetime('2018-12-05'),   pd.to\_datetime('2018-12-10') ]) youtube[filters] |

Como pôde ver nos exemplos acima, o .isin é muito útil e pode ser utilizado em colunas do tipo texto, número e data.

Em vez de usar o operador | para criar uma condição múltipla, basta utilizar o.isin e passar os valores que deseja aceitar por meio de uma lista.

Explicando melhor, substituímos isso:

|  |
| --- |
| (youtube['canal'] == '3Blue1Brown') | (youtube['canal'] == 'TED') | (youtube['canal'] == 'Google') |

Por isso:

|  |
| --- |
| youtube['canal'].isin(['3Blue1Brown', 'TED', 'Google']) |

Bem melhor!

Um outro caso comum é com faixas de números ou datas. Nós podemos utilizar a função .between nesses casos.

Registros de vídeos com comentários entre 1.000 a 2.000:

|  |
| --- |
| filters = youtube['quantidade\_de\_comentarios'].between(1000, 2000) youtube[filters] |

Registros de vídeos com comentários entre 999 a 1.999:

|  |
| --- |
| filters = youtube['quantidade\_de\_comentarios'].between(1000, 2000, inclusive=**False**) youtube[filters] |

Registros de vídeos publicados entre 27/02/2018 e 02/03/2028:

|  |
| --- |
| filters = youtube['data\_da\_observacao'].between(  pd.to\_datetime('2018-02-27'),   pd.to\_datetime('2018-03-02') ) youtube[filters] |

Registros de vídeos publicados entre abril e agosto:

|  |
| --- |
| filters = youtube\_clean['data\_da\_observacao'].dt.month.between(4, 8) youtube[filters] |

Dessa forma, evitamos o uso do &. Substituímos:

|  |
| --- |
| (youtube['quantidade\_de\_comentarios'] >= 1000) & (youtube['quantidade\_de\_comentarios'] <= 2000) |

Por:

|  |
| --- |
| youtube['quantidade\_de\_comentarios'].between(1000, 2000) |

Bem melhor!

Dê preferência ao uso das funções .isin e .between sempre que for possível. Em outras situações, você realmente vai precisar utilizar & e |,normalmente quando a condição é feita utilizando mais que 1 coluna.

Atenção a um detalhe importante: ao montar as condições em uma só linha, utiliza () para delimitar cada condição. Perceba, nos exemplos abaixo, como isso foi feito.

Registros de vídeos onde o nome do canal é "LastWeekTonight" e a quantidade de comentários é maior que 100:

|  |
| --- |
| filters = (youtube['canal'] == 'LastWeekTonight') & (youtube['quantidade\_de\_comentarios'] > 100) youtube[filters] |

Registros de vídeos com 1.000, 2.000 ou 3.000 comentários:

|  |
| --- |
| filters = (youtube['quantidade\_de\_comentarios'] == 1000) | (youtube['quantidade\_de\_comentarios'] == 2000) | (youtube['quantidade\_de\_comentarios'] == 3000) youtube[filters] |

Existe também uma outra forma, que não exige o uso dos (), de fazer esse filtro com múltiplas condições. Veja abaixo como isso é feito:

|  |
| --- |
| filter\_1 = youtube['canal'] == 'LastWeekTonight' filter\_2 = youtube['quantidade\_de\_comentarios'] > 100  youtube[filter\_1 & filter\_2]  # utilizando o comparador “or”:  # youtube[filter\_1 | filter\_2] |

Assim as condições ficam mais organizadas, legíveis e podem se reutilizadas!

Um detalhe, que optei por deixar para o fim e espero que não fique chateado, é que você pode inserir a condição dentro do filtro, sem a necessidade de criar a condição primeiro.

Veja alguns exemplos:

|  |
| --- |
| youtube[youtube['canal'] == 'LastWeekTonight']  youtube[(youtube['canal'] == 'LastWeekTonight') & (youtube['quantidade\_de\_comentarios'] > 100)] |

Vamos fazer uma lista de prós e contras dessas opções.

Vantagens:

* Economia de linhas de código

Desvantagens:

* Legibilidade: fica mais difícil de ler
* Flexibilidade: há um trabalho maior em adicionar/remover condições
* Reutilização: ao usar o mesmo filtro em outras partes do código, vai precisar escrever a condição novamente

Aconselhamos fortemente que utilize a segunda opção.

##### Outros filtros

Para finalizar, deixaremos mais dois exemplos um pouco "avulsos", porém importantes.

Exemplo 1 - Como filtrar linhas que têm pelo menos 1 coluna com valor Null:

|  |
| --- |
| filters = youtube.isnull().any(axis=1) youtube[filters] |

Entenda as etapas:

1. Aplicamos .isnull no Dataframe inteiro, ou seja, em todas as colunas. Isso retorna valores booleanos para cada célula do Dataframe
2. Verificamos se pelo menos 1 das colunas (.any) contém um valor Null
3. Obtemos um único resultado por linha (axis=1 indica para verificar por linha)
4. Aplicamos o filtro da forma que já estamos acostumados

Exemplo 2 - Como inverter o filtro (o contrário do que fizemos no Exemplo 1):

|  |
| --- |
| filters = youtube.isnull().any(axis=1) youtube[~filters] |

Resumindo, basta adicionar ~ para inverter o filtro.

#### Groupby + Aggregate

A função .groupby serve para agrupar informações de um Dataframe de acordo com os valores de 1 ou mais colunas.

Após o agrupamento, normalmente utilizamos a função .agg para resumir os valores de cada grupo, informando quais funções queremos utilizar.

Vamos ver um exemplo utilizando o dataset do YouTube:

**Objetivo:** Quantos vídeos e quantos likes cada canal do dataset tem?

**Etapas:**

1. Utilizamos a função .groupby na coluna "canal", criando um grupo de informação (com os registros) de cada canal
2. Aplicamos a função pd.Series.nunique na coluna "vídeo" e a função .sum na coluna "gostei"

O código para fazer isso é:

|  |
| --- |
| youtube.groupby('canal').agg({  'video': pd.Series.nunique,  'gostei': sum  }) |



Como pode ver, é uma tarefa simples de ser feita com o Pandas.

Na sequência, vou explicar um pouco melhor como é esse processo e também mostrar algumas técnicas úteis com base em problemas que podemos encontrar no dia a dia.

##### Agrupamentos

É importante entender como o agrupamento do Pandas funciona. Isso facilita o entendimento do que está sendo feito e como chegar ao resultado que deseja.

Aproveitando o exemplo anterior, vamos agrupar as informações pela coluna "canal":

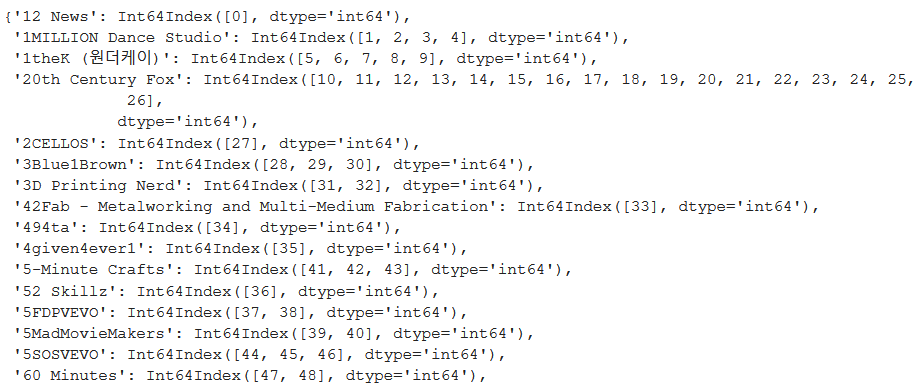
|  |
| --- |
| youtube.groupby('canal') |

|  |
| --- |
| <pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f6bda748> |

Essa função retorna um objeto **pandas.core.groupby.generic.DataFrameGroupBy**.

Nesse momento, um comando muito esclarecedor que você pode utilizar é:

|  |
| --- |
| youtube.groupby('canal').groups |

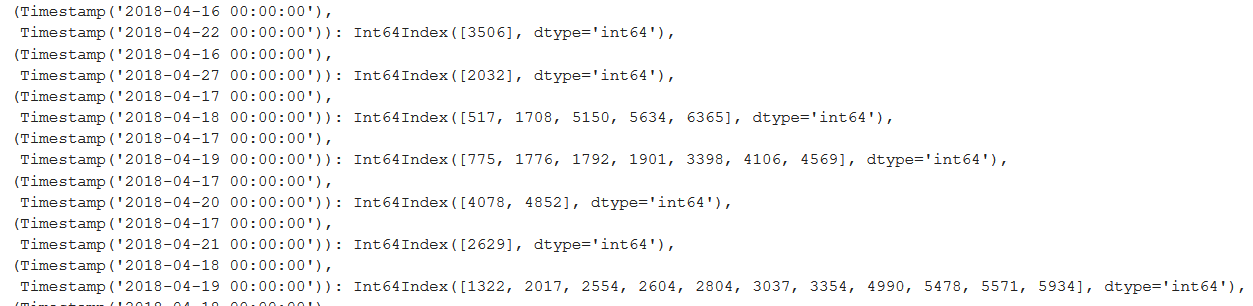


Esse comando retorna todos os grupos que foram criados com base no que foi solicitado (no nosso exemplo, grupos de acordo com a coluna "canal").

Perceba que é um dicionário onde cada item é um grupo onde o valor é uma lista com os índices que fazem parte daquele grupo. Dessa forma o Pandas armazena os grupos e as informações de cada grupo.

Você também pode criar grupos de acordo com valores de múltiplas colunas, basta informá-las por meio de uma lista:

|  |
| --- |
| youtube.groupby(['data\_da\_publicacao\_do\_video', 'data\_da\_observacao']).groups |



Assim, todos os diferentes vídeos que tenham sido publicados e observados em dias em comum estarão juntos.

Os agrupamentos são úteis, mas representam apenas uma parte do objetivo. Depois de agrupar, precisamos realizar alguma ação.

Na próxima seção vamos falar sobre agregação e ver como podemos aplicá-las nos grupos que acabamos de criar.

##### Agregação

Para fazer agregações, precisamos informar quais funções queremos aplicar e em quais colunas.

Isso cria algumas combinações (função x coluna) diferentes e na sequência vamos ver alguns exemplos de cada uma dessas combinações.

Antes de tudo, vamos alinhar as possibilidades:

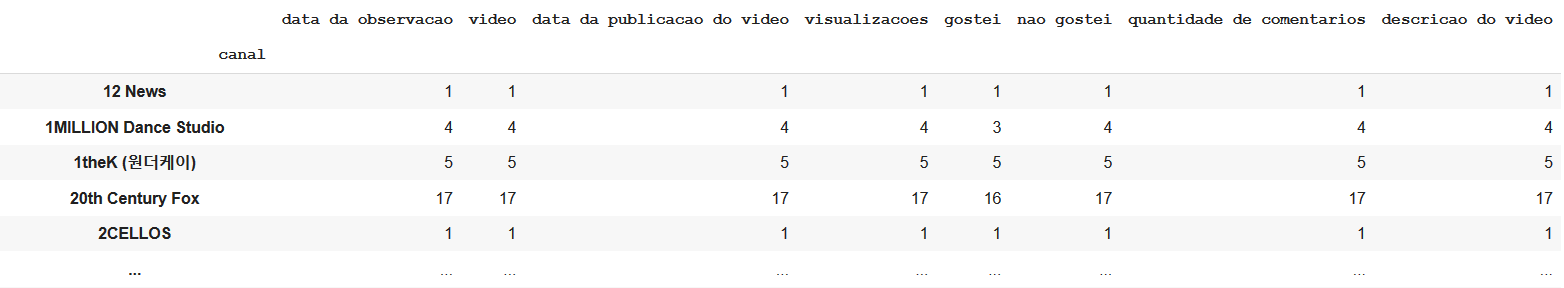
* Para as colunas, considere as colunas que já conhece do dataset do YouTube.
* Para as funções, seguem algumas muito utilizadas e úteis: count, sum, mean, median, min, max, std, cumsum, pd.Series.nunique

**Combinação "1 função x todas as colunas"**

Não é muito comum, mas você pode aplicar essas funções diretamente no objeto DataFrameGroupBy. Dessa forma, a função será aplicada em todas (ou quase todas, dependendo da função) as colunas.

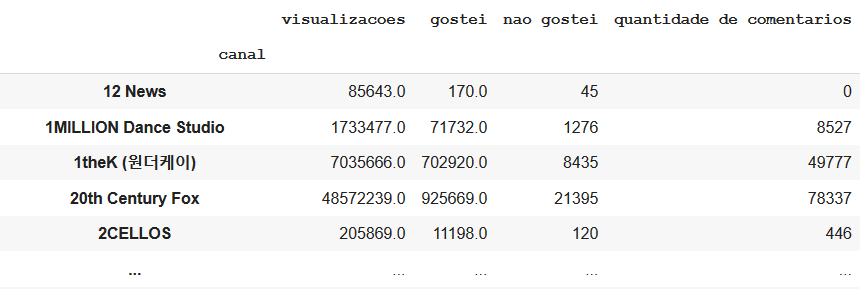
.count (contagem das células que não tem valores NA - aplicado em todas as colunas):

|  |
| --- |
| youtube.groupby('canal').count() |



.sum (soma - aplicado apenas nas colunas númericas):

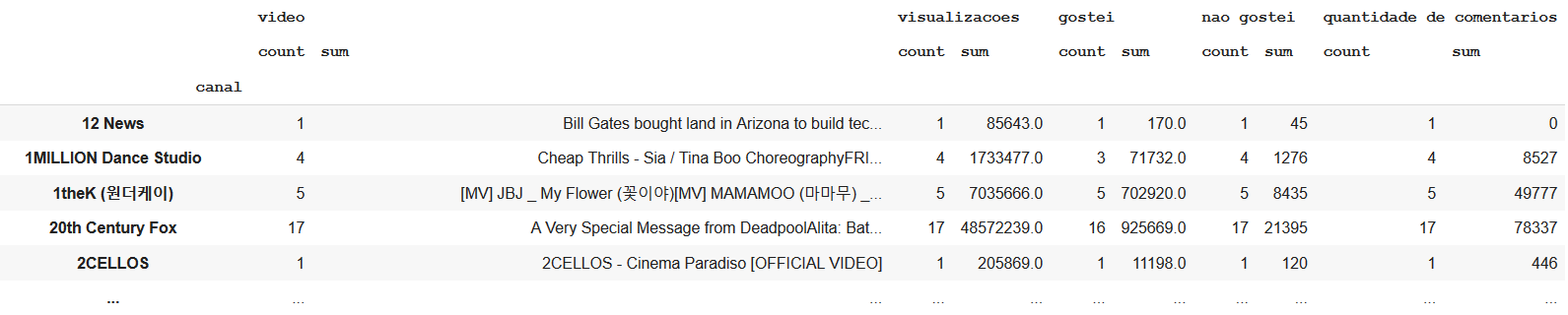
|  |
| --- |
| youtube.groupby('canal').sum() |



**Combinação "várias funções x todas as colunas"**

Para utilizar mais de 1 função de agregação, você vai precisar da função .agg. Basta informar uma lista com as funções que quer utilizar:

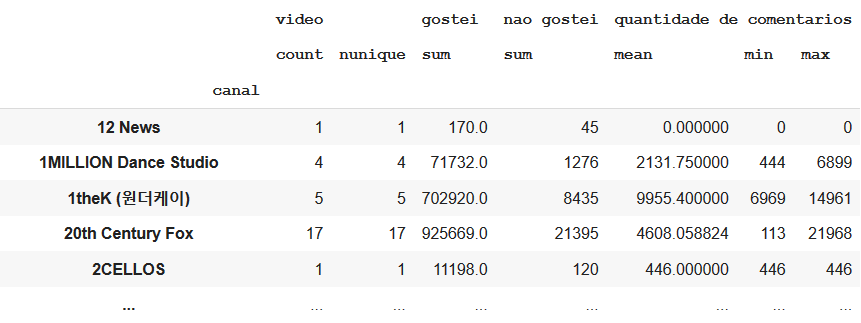
|  |
| --- |
| youtube.groupby('canal').agg(['count', 'sum']) |



Combinação "1 ou mais funções x algumas colunas"

Existem algumas formas diferentes de se fazer isso e aqui vamos ver a mais flexível:

|  |
| --- |
| youtube.groupby('canal').agg({  'video': ['count', pd.Series.nunique],  'gostei': 'sum',  'nao\_gostei': 'sum',  'quantidade\_de\_comentarios': ['mean', 'min', 'max'] }) |



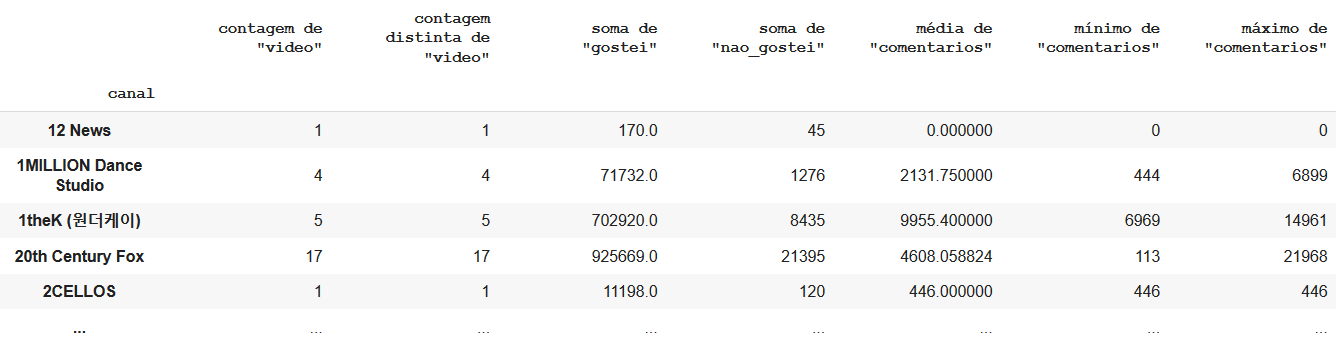
Essa alternativa é muito flexível porque permite que uma ou mais funções sejam aplicadas para cada coluna desejada. É preciso somente passar um dicionário onde o nome da chave é o nome da coluna e o valor seja as funções a serem aplicadas.

Um detalhe dessa forma de realizar a agregação é o seguinte: é ótimo para visualizar, difícil para utilizar.

Perceba que ambas as colunas "gostei" e "nao\_gostei" ficam com o título "sum", porque foi essa a função aplicada nessas colunas, mas essa repetição não é muito interessante quando queremos selecionar uma dessas colunas.

Uma forma muito interessante de resolver isso é a seguinte:

|  |
| --- |
| youtube.groupby('canal').agg(\*\*{  'contagem de "video"': ('video', 'count'),  'contagem distinta de "video"': ('video', pd.Series.nunique),  'soma de "gostei"': ('gostei', 'sum'),  'soma de "nao\_gostei"': ('nao\_gostei', 'sum'),  'média de "comentarios"': ('quantidade\_de\_comentarios', 'mean'),  'mínimo de "comentarios"': ('quantidade\_de\_comentarios', 'min'),  'máximo de "comentarios"': ('quantidade\_de\_comentarios', 'max'), }) |



A sintaxe é parecida. O que muda é:

* Coloque \*\* antes do dicionário com as informações
* A chave do dicionário é o nome que deseja para a coluna
* O valor é uma tupla com o nome da coluna e a função a ser aplicada

Agora você já tem o conhecimento necessário para realizar agrupamentos e agregações em seus datasets!

### **Tratamento de Nulos**

Colunas com altos índices de nulos são um problema, seja para estudos exploratórios, seja para modelos de Machine Learning pois alteram a proporção e informações estatísticas como média, moda e medianas, possibilitando erros em nossa análise.

|  |
| --- |
| youtube.isnull().sum() |

|  |
| --- |
| data\_da\_observacao 0 video 0 canal 0 data\_da\_publicacao\_do\_video 0 visualizacoes 6 gostei 11 nao\_gostei 0 quantidade\_de\_comentarios 0 descricao\_do\_video 107 dtype: int64 |

Em nosso dataset, temos poucos casos de dados nulos, 6 para Visualizações, 11 para Gostei e 107 para Descrição. Para resolvermos esse problema temos à disposição diversas técnicas que veremos abaixo

#### Exclusão de dados nulos

Quando temos baixo índice de nulos podemos removê-los sem afetar nossa análise

|  |
| --- |
| youtube = youtube.dropna() |

|  |
| --- |
| youtube.isnull().sum() |

|  |
| --- |
| data\_da\_observacao 0 video 0 canal 0 data\_da\_publicacao\_do\_video 0 visualizacoes 0 gostei 0 nao\_gostei 0 quantidade\_de\_comentarios 0 descricao\_do\_video 0 dtype: int64 |

#### Uso de medidas de tendência central

Quando temos uma grande quantidade de dados com nulos, podemos utilizar o recurso da média, mediana e moda. Dependendo da nossa distribuição e tipo do dado, utilizaremos um medidas específicas

* Para dados categóricos podemos utilizar a moda, ou seja, a observação com maior repetição.
* Quando o desvio padrão dos meus dados são altos podemos utilizar a mediana, utilizando assim, a observação que separa a distribuição ao meio (50%).
* Quando temos dados quantitativos dentro da normalidade, podemos utilizar a média.

Em nosso caso, temos poucas observações nulas e seu resultado esperado é quantitativo, ou seja, descartaremos a opção de utilizarmos a moda.

Como exemplo, utilizaremos os nulos da coluna Gostei, deste modo, filtraremos todas as observações na qual a coluna Gostei é nula e exibiremos somente os 5 primeiros ítens

|  |
| --- |
| youtube[youtube['gostei'].isnull()].head() |

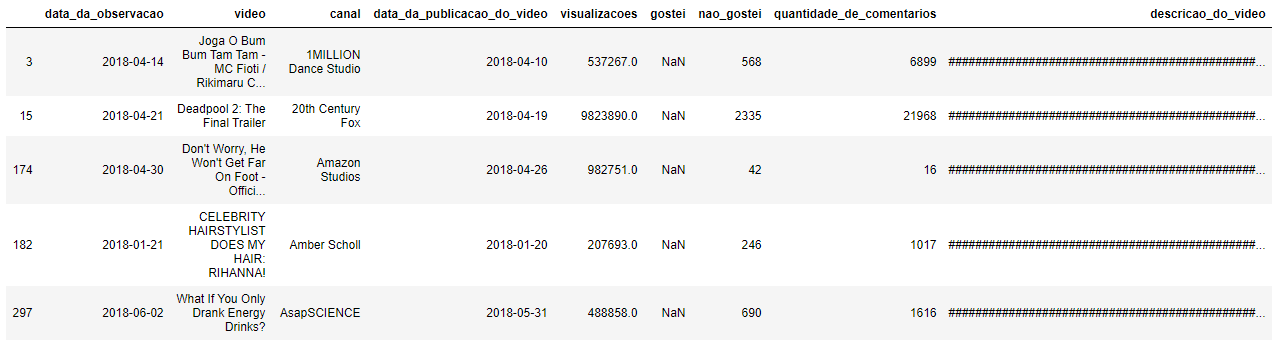


Figura XXX - Dataset filtrado para os ítens nulos

Para este exemplo iremos canal a canal observar seus dados, iniciaremos nossa jornada de tratamento pela 20th Century Fox.

|  |
| --- |
| youtube[youtube['canal'] == '20th Century Fox'] |

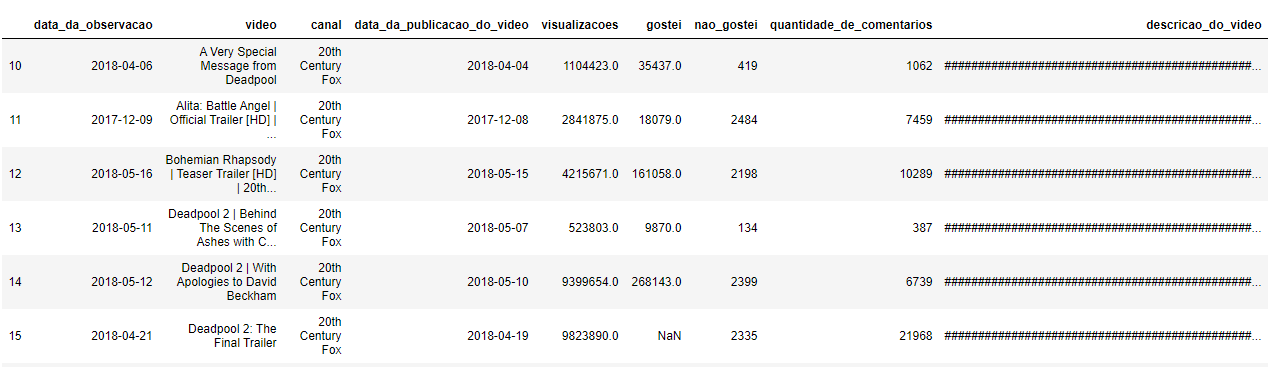


Figura XXX - Dataset filtrado para o canal 20th Century Fox

Observamos que o filme Deadpool apresenta um dado nulo, contudo, temos outras observações do mesmo filme que podemos utilizar como base de cálculo. Filtraremos então, todos os filmes Deadpool da 20th Century Fox

|  |
| --- |
| youtube[  (youtube['video'].str.contains('Deadpool')) &   (youtube['canal'] == '20th Century Fox') ] |

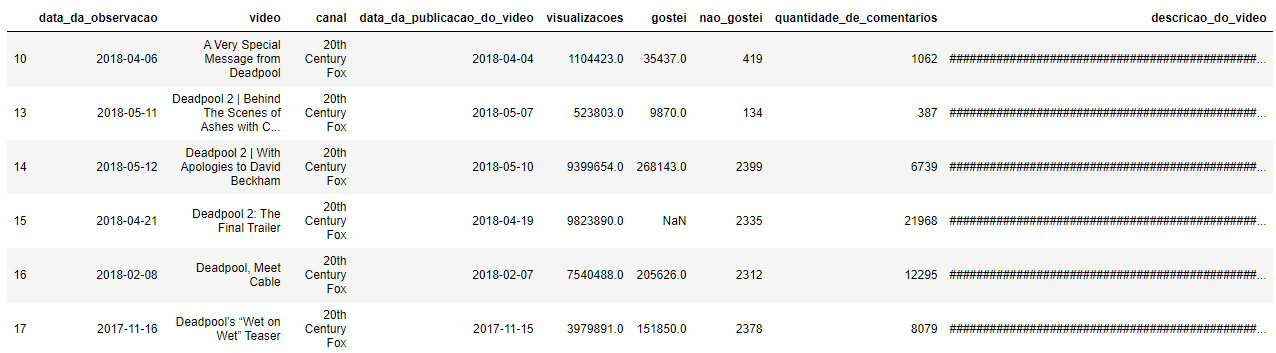


Figura XXX - Dataset filtrado para o canal 20th Century Fox e Título Deadpool

Para realizar o filtro acima, utilizamos a função str.contains na qual nos retorna todos os ítens que contém uma string, em nosso caso, Deadpool, contudo, mantivemos o filtro de Canal para nos mostrar a 20th Century Fox, utilizamos o operador & para realizar tal ação

Vamos agora calcular a média de “Gostei” e aplicar à nossa opção nula. Existem diversas formas para executar esta ação, por questões didáticas, a veremos passo à passo

|  |
| --- |
| media = youtube[  (youtube['video'].str.contains('Deadpool')) &   (youtube['canal'] == '20th Century Fox') &   (youtube['gostei'].notnull()) ]['gostei'].mean() |

Calculamos a média de “Gostei” de todos os ítens que contém no Título o texto “Deadpool”, sejam da 20th Century Fox e ao mesmo tempo não nulos.

|  |
| --- |
| media 134185.2 |

Vamos agora substituir o nulo pelo valor da média por meio da função fillna()

|  |
| --- |
| youtube[  (youtube['video'].str.contains('Deadpool')) &   (youtube['canal'] == '20th Century Fox') ].fillna(media) |



Figura XXX - Dataset com nulo substituído pela média

#### Preenchendo nulos com dados padrão

Uma outra forma de lidarmos com nulos é preenchermos com dados padrão e para isso utilizamos o comando fillna(). Em nosso caso, temos 107 observações com a descrição nula, e queremos substituir pelo termo N/A

|  |
| --- |
| youtube['descricao\_do\_video'] = youtube['descricao\_do\_video'].fillna('N/A') |

|  |
| --- |
| youtube\_clean.isnull().sum() |

|  |
| --- |
| data\_da\_observacao 0 video 0 canal 0 data\_da\_publicacao\_do\_video 0 visualizacoes 6 gostei 11 nao\_gostei 0 quantidade\_de\_comentarios 0 **descricao\_do\_video 0** dtype: int64 |

A função fillna() ainda possui outros métodos de preenchimento como o ffill e o bfill. O método ffill preenche com os dados de cima para baixo, ou seja, preenche todos os nulos com o valor da primeira ocorrência não nula acima.

|  |
| --- |
| youtube[youtube['canal'] == 'The View'] |

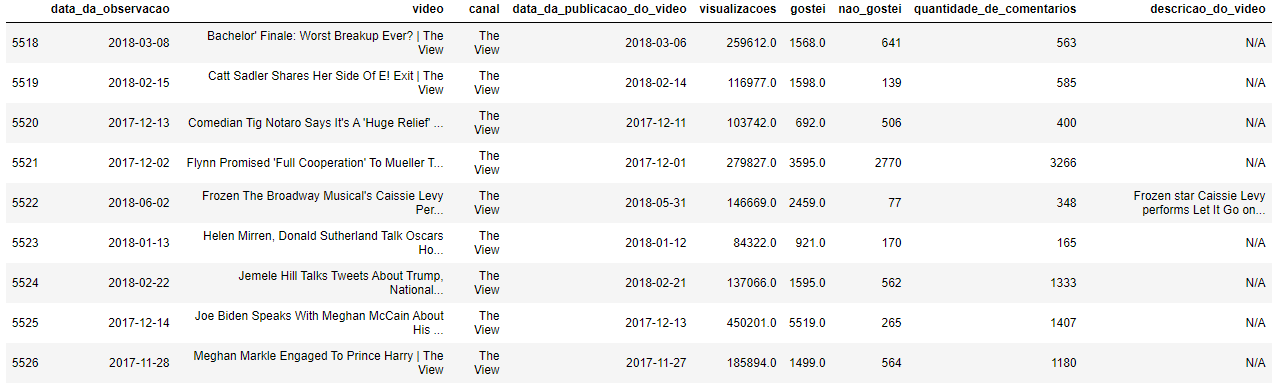


Figura XXX - Dataset com nulos na descrição

|  |
| --- |
| youtube[youtube['canal'] == 'The View'].fillna(method='ffill') |

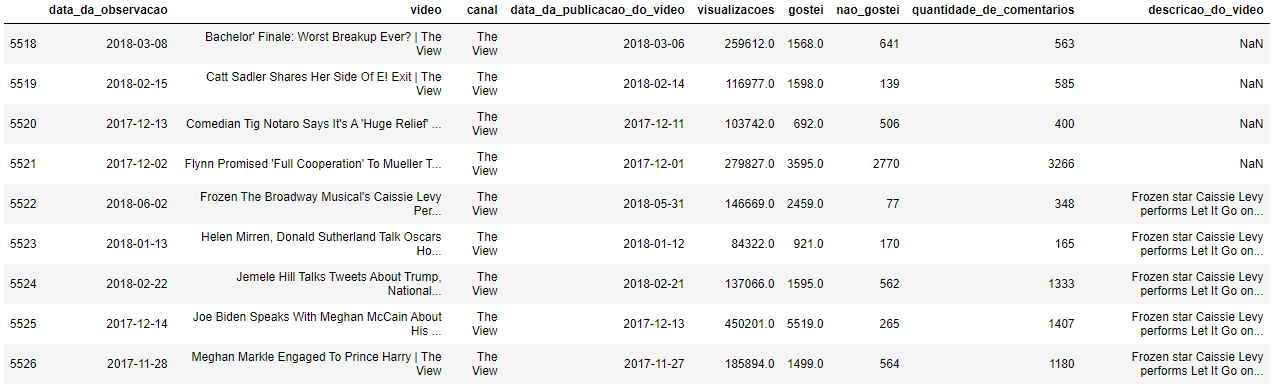


Figura XXX - Dataset após a operação de fillna com método ffill

Já o inverso ocorre com o método bfill:

|  |
| --- |
| youtube[youtube\_clean['canal'] == 'The View'].fillna(method='bfill') |

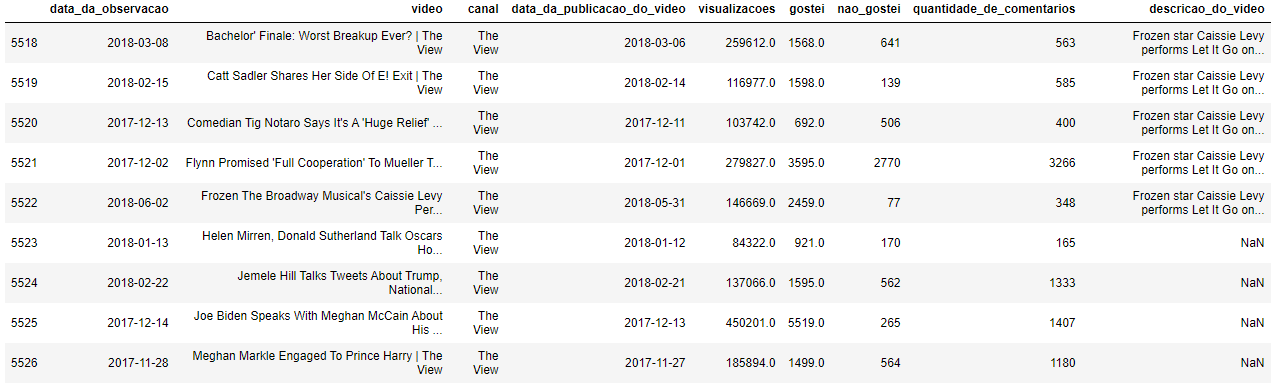


Figura XXX - Dataset após a operação de fillna com método bfill

#### Utilizando Machine Learning

Outro método é a utilização de Machine Learning para que a máquina possa prever os resultados vazios e com isso possamos a trabalhar com nosso dataset sem maiores problemas. Veremos mais detalhes sobre Machine Learning e predição de dados no Capítulo 62.