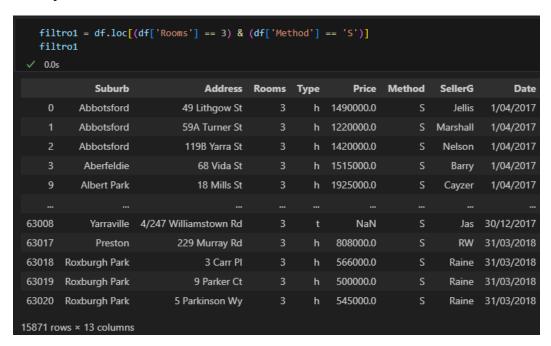
Aprofundando em filtros com Pandas

Filtrando dados com condições

Já vimos anteriormente como filtrar dados com condições através da propriedade de indexação **loc[]**!! E agora vamos dar uma reforçada como fazemos filtros com **N condições** diferentes!



Veja como é simples!! Basta separar cada condição do filtro, que seria cada listagem do objeto **Series Boolean (máscara booleana)**, através de **()**.

Filtrando com a função contains()

Em alguns momentos pode ser que precisamos realizar algum filtro referente a busca de uma **parte do texto**, como por exemplo filtrar dados que contém a palavra **streat (st)** ou **rua**.

E para isso no momento do filtro, primeiramente chamamos a propriedade **str**, para que cada dado do nosso objeto **series** seja convertido em **string**, e logo em seguida utilizamos a função **contains()**, como podemos ver a seguir:

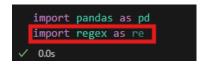
<pre>filtro2 = df.loc[df['Address'].str.contains('St')] filtro2</pre>									
′ 0.0s									
	Suburb	Address	Rooms	Туре	Price	Method	SellerG	Date	Post
0	Abbotsford	49 Lithgow St	3	h	1490000.0	s	Jellis	1/04/2017	
1	Abbotsford	59A Turner St	3	h	1220000.0	s	Marshall	1/04/2017	
2	Abbotsford	119B Yarra St	3	h	1420000.0	S	Nelson	1/04/2017	
3	Aberfeldie	68 Vida St	3	h	1515000.0	S	Barry	1/04/2017	
	A:	4/22 FI C+	2		F20000 0	c	1-11:-	1/04/2017	

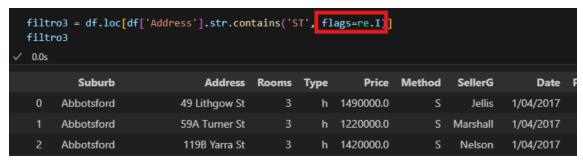
Utilizando regular expression

Vimos que anteriormente conseguimos realizar filtro referente a busca de uma parte do texto utilizado a função **contains()**. Porém da forma como está, caso o texto que estivemos filtrando tiver em **uper case** ou em **lower case** diferente do texto informado do filtro, logo o filtro não será realizado!

E para resolver podemos utilizar o recurso de **regular expression (regex)**, pois com o **regex** conseguimos descrever padrões de texto! Muito utilizado para realizar **validações dentro de um texto** ou até mesmo **extrair parte de um texto**!

E para usar o **regular expression** nesse caso, na própria função do **contains()**, temos um parâmetro chamado **flags**, onde podemos definir nossa expressão regular nele!

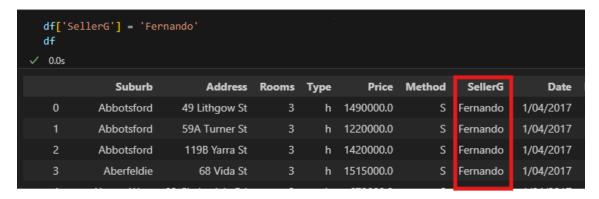




Veja que primeiramente foi importado a biblioteca do **regex**, e após importar a biblioteca utilizamos a propriedade **I** da biblioteca no parâmetro **flags** do **contains()**, onde estamos definindo que seja ignorado letras **maiúsculas** e **minúscula** no filtro!

Modificando dados no DataFrame

Já vimos anteriormente como podemos atualizar dados de um **DataFrame**, ou de um objeto **Series** pertencente a um **DataFrame**, como podemos ver a seguir:



Veja que alteramos todos os valores da coluna (**Series**) '**SellerG**' para '**Fernando**'! Porém não queremos alterar todos os dados dessa coluna para **Fernando**!!

Dito isso, queremos alterar somente o nome de **Nelson** para **Fernando**, logo precisaremos realizar um filtro primeiro para pegar todos os registros referente ao **SellerG** de **Nelson**, só para depois realizar a atualização em si para **Fernando**!



Porém se observar no resultado, temos um alerta que o **Jupyter** nos apresenta, dizendo que estamos tentando modificar uma parte (**slice**) de um **DataFrame**, porém o **Pandas** não sabe se estamos tentando modificar a lista original (**df**) ou uma cópia que foi feito através do **slice**, quando realizamos o filtro e armazenamos esse **slice** através da variável (**df_update**)!!

E para resolver isso é muito simples!! Basta no momento do filtro que usamos através da propriedade **loc[]**, definirmos o valor a ser modificado no mesmo comando, assim o **Pandas** saberá que a modificação deve ser feita no **DataFrame** original (**df**)!!



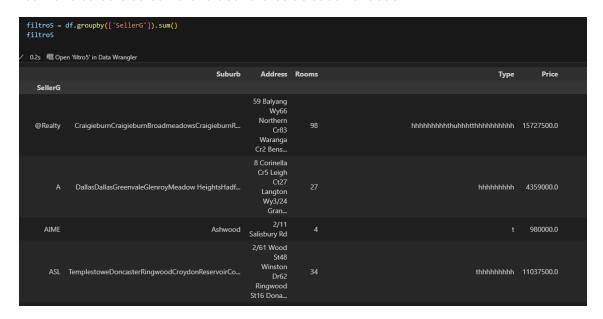
Obs: Note que precisamos definir qual coluna(s) queremos modificar, pois da forma padrão o filtro retornara o DataFrame filtrado com todas as colunas, e nesse caso não queremos modificar o DataFrame inteiro e sim apenas uma coluna apenas (SellerG)!

```
df.loc[df['SellerG'] == 'Nelson', ['SellerG']] = 'Fernando'
```

Agrupando dados com GroupBy

Um recurso bem importante que podemos utilizar também, seria a utilização do **GroupBy!!** Onde basicamente irá agrupar todos os nossos dados através de uma coluna de referência!

Como por exemplo, vamos agrupar os nossos dados calculando a soma dos valores referente a cada vendedor (**SallerG**)!! Dito isso, usaremos a função **groupby()**, passando o **SallerG** como parâmetro, e consequentemente chamando também a função **sum()** para realizar o cálculo da somatória dos valores de cada vendedor!



Veja que foi listado corretamente, **somando todos os valores agrupados** para cada vendedor (**SellerG**)!!

Porém por padrão, está sendo ordenado referente aos vendedores (**SellerG**) e queremos que seja ordenado de forma decrescente os preços (**Price**), ou seja, que seja ordenado os vendedores que tiveram as maiores vendas!

