

# Pandas: Processo de Análise

Pandas é uma poderosa biblioteca de **análise e manipulação de dados de código aberto**. Ele pode ajudá-lo a fazer várias operações sobre os dados e gerar diferentes relatórios sobre eles. Vou dividir isso em dois artigos...

1. Básico, que eu vou cobrir nesta história. Vou cobrir funções básicas pandas que lhe darão uma visão geral de como você pode começar a trabalhar com Pandas e como ele pode ajudá-lo a economizar muito do seu tempo.
2. Avançado, através de funções avançadas que facilita a resolução de problemas de análise complexos. Abrange temas como estilo, plotagem, tabelas pivôs, etc.

Até o final d artigo, seremos capazes de responder a seguir:

1. Como ler dados de diferentes formatos.
2. Exploração de dados usando diferentes APIs.
3. Quais são as dimensões de Dados.
4. Como encontrar o Resumo de Dados.
5. Como verificar diferentes estatísticas sobre Dados.
6. Como selecionar o subconjunto de Dados.
7. Um forro para contar valores únicos com frequências de cada valor.
8. Como filtrar os dados com base nas condições.
9. Salvar dados em diferentes formatos.
0. Aplicando várias operações usando encadeamento.

Antes de começar, certifique-se de ter instalado Pandas. Se não, usar o seguinte comando para baixá-lo.

```
# If you are using Anaconda Distribution (recommended)
conda install -c conda-forge pandas

# install pandas using pip
pip install pandas

# import pandas in notebook or python script
import pandas as pd
```

## 1. Ler dados: read\_csv ; read\_excel ; read\_json

O ponto de partida de qualquer análise de dados é a aquisição do conjunto de dados. Pandas fornece diferentes funções para ler os dados de diferentes formatos. Os mais usados são –

### read\_csv()

Isso permite que você leia um arquivo CSV.

```
pd.read_csv('path_to_your_csv_file.csv')
```

### Ler arquivos em Pandas

Os pandas fornecem diferentes opções para configurar nomes de colunas ou tipos de dados ou o número de linhas que você gostaria de ler. Confira os Pandas read\_csv API para mais detalhes:

```
pandas.read_csv - pandas 1.2.0 documentation
pandas.read_csv( filepath_or_buffer, 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,
📄 https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\_csv.html
```

### read\_json()

Esta API da Pandas ajuda a ler dados JSON e funciona muito bem para dados já achatados. Vamos dar um exemplo JSON para ver como podemos convertê-lo em uma mesa plana.

```
# Sample Record: Flattend JSON i.e. no nested array or dictionary
json_data = {
    "Scaler": "Standard",
    "family_min_samples_percentage": 5,
    "original_number_of_clusters": 4,
    "eps_value": 0.1,
    "min_samples": 5,
    "number_of_clusters": 9,
    "number_of_noise_samples": 72,
}
# reading JSON data
pd.read_json(json_data)
```

Apenas lendo o JSON converteu-o em uma mesa plana abaixo.

	Scaler	family_min_samples_percentage	original_number_of_clusters	eps_value	min_samples	number_of_clusters	number_of_noise_sai
0	Standard	5	4	0.1	5	9	72
1	Standard	5	4	0.1	10	6	89


## json\_normalize() para dados semiestruturados

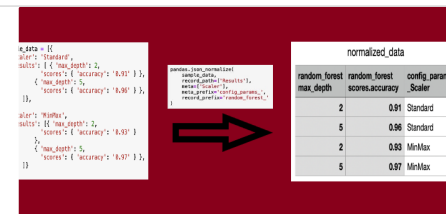
Não funciona bem quando os dados JSON são semiestruturados, ou seja, contém lista aninhada ou dicionários. Pandas fornecem uma API

**json\_normalize** para isso também se você quiser aprender mais, confira:

How to parse JSON data with Python Pandas?

If you are doing anything related to data whether it is Data Engineering, Data Analytics, or even Data Science, you would have surely come across JSONs. JSON (JavaScript Object Notation) is

 <https://towardsdatascience.com/how-to-parse-json-data-with-python-pandas-f84fbd0b1025>



## Tipos de dados que Pandas suporta

Existem muitos outros tipos de dados que pandas suporta. Verificar a documentação do Pandas quando usarmos outros tipos de dados.

Input/output - pandas 1.2.0 documentation

Load pickled pandas object (or any object) from file.

 <https://pandas.pydata.org/pandas-docs/stable/reference/io.html>

Lendo o conjunto de dados do Titanic que vamos usar aqui usando o comando `read_csv()`:

```
# Loading Titanic Dataset into titanic_data variable
titanic_data = pd.read_csv('titanic_train.csv')
```

Isso criará um DataFrame pandas (como tabelas) e o armazenará na variável `titanic_data`.

Em seguida, veremos como obter mais detalhes sobre os dados que carregamos.

## 2. Explorar dados

Uma vez que tenhamos carregado os dados, gostaríamos de revisá-los. Pandas fornece diferentes APIs que podemos usar para explorar dados.

### head( )

Isso é como um comando TOP no SQL e nos dá 'n' registros desde o início do DataFrame.

```
# Selecting top 5 (n=5) records from the DataFrame
titanic_data.head(5)
```

	PassengerId	Survived	Pclass	Name	Sex
0	1	0	3	Braund, Mr. Owen Harris	male
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female
2	3	1	3	Heikkinen, Miss. Laina	female
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female
4	5	0	3	Allen, Mr. William Henry	male

### tail( )

Isso nos dá os registros 'n' do final do DataFrame.

```
# Selecting last 5 (n=5) records from the DataFrame
titanic_data.tail(5)
```

	PassengerId	Survived	Pclass	Name	Sex
886	887	0	2	Montvila, Rev. Juozas	male
887	888	1	1	Graham, Miss. Margaret Edith	female
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female
889	890	1	1	Behr, Mr. Karl Howell	male
890	891	0	3	Dooley, Mr. Patrick	male

## sample()

Isso capta aleatoriamente o número 'n' de registros dos dados. Nota: a saída deste comando pode ser diferente em diferentes corridas.

```
titanic_data.sample(5)
```

	PassengerId	Survived	Pclass	Name	Sex
691	692	1	3	Karun, Miss. Manca	female
470	471	0	3	Keefe, Mr. Arthur	male
357	358	0	2	Funk, Miss. Annie Clemmer	female
773	774	0	3	Elias, Mr. Dibo	male
808	809	0	2	Meyer, Mr. August	male

## 3. Dimensões de dados usando a forma

Uma vez que temos os dados, precisamos saber quantas linhas ou colunas estamos lidando:

## shape()

```
# shape of the dataframe, note there is no parenthesis at the end as it is a property of dataframe
titanic_data.shape
(891, 12)
```

(891, 12) significa que temos 891 linhas e 12 colunas em nosso conjunto de dados titanic.

## 4. Resumo de Dados utilizando info( )

Vamos ver a saída deste primeiro:

### info()

```
titanic_data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  -
0   PassengerId     891 non-null    int64
1   Survived        891 non-null    int64
2   Pclass         891 non-null    int64
3   Name            891 non-null    object
4   Sex             891 non-null    object
5   Age            714 non-null    float64
6   SibSp          891 non-null    int64
7   Parch          891 non-null    int64
8   Ticket         891 non-null    object
9   Fare           891 non-null    float64
10  Cabin          204 non-null    object
11  Embarked       889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB

```

Como você pode ver 'info' fornece um bom resumo dos dados que temos, vamos entendê-los um por um.

1. Detalhes do índice- cada dataframe em Pandas tem um índice que, basicamente, se você está familiarizado com SQL, é como um índice que criamos para acessar os dados. Aqui significa que temos um RangeIndex de 0 a 890, ou seja, um total de 891 linhas.
2. Cada linha na tabela gerada por 'info' nos dá os detalhes sobre a coluna que temos, o número de valores que estão lá na coluna e o tipo de dados que os pandas atribuíram. Isso é útil para ter um vislumbre de dados perdidos como podemos dizer que temos dados 'Age' apenas para 714 linhas.
3. Uso da memória- Pandas carrega o quadro de dados na memória e isso nos diz quanta memória nosso conjunto de dados está usando. Isso é útil

quando temos grandes conjuntos de dados e, portanto, os pandas têm uma API específica 'memory\_usage' para mais opções.

## 5. Estatísticas de dados usando describe( )

### describe()

Isso nos dá as estatísticas sobre o conjunto de dados. Como você vê para o nosso dataframe abaixo ele mostra:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Isso nos dá muitas informações para cada coluna, como contagem de registros (não conta registros ausentes como você pode ver em Age), média, desvio padrão, percentuais mínimos e diferentes quânticos.

Por padrão, este comando fornece informações sobre tipos de dados numéricos como int ou float. Para obter estatísticas sobre nossas colunas de "objeto", podemos executar... Como você pode ver, isso nos dá muitas informações para cada coluna, como contagem de registros (não conta registros ausentes como você pode ver em Age), média, desvio padrão, percentuais mínimos e diferentes quânticos.

Por padrão, este comando fornece informações sobre tipos de dados numéricos como int ou float. Para obter estatísticas sobre nossas colunas de "objeto", podemos executar...

### describe(include=['O'])

```
# show stats about object columns
titanic_data.describe(include=['O'])
```



	Name	Sex	Ticket	Cabin	Embarked
<b>count</b>	891	891	891	204	889
<b>unique</b>	891	2	681	147	3
<b>top</b>	Panula, Master. Eino Viljami	male	1601	C23 C25 C27	S
<b>freq</b>	1	577	7	4	644

Adicionamos parâmetro 'incluir' para descrever que é uma lista de objetos, podemos passar vários valores como...

- `include = ['O', 'int64']` – dará estatísticas sobre colunas do tipo Object e Int64 no DataFrame.
- `include = ['O', 'float64']` – dará estatísticas sobre colunas do tipo Object e float64 no DataFrame.

## 6. Seleção de dados usando loc e iloc

São funções muito úteis para nos ajudar na seleção de dados. Usando estes, podemos selecionar qualquer parte dos dados. Para entendê-lo melhor, vamos alterar o índice dos dados.

### `set_index()`

Definindo um índice para o dataset.

```
# Change index of our DataFrame from RangeIndex to 'Ticket' values
titanic_ticket_index = titanic_data.set_index('Ticket')
```

```
titanic_ticket_index.head(5)
```

executed in 12ms, finished 15:37:37 2020-07-31

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
Ticket										
A/5 21171	1	0	3	male	22.0	1	0	7.2500	NaN	S
PC 17599	2	1	1	female	38.0	1	0	71.2833	C85	C
STON/O2. 3101282	3	1	3	female	26.0	0	0	7.9250	NaN	S
113803	4	1	1	female	35.0	1	0	53.1000	C123	S
373450	5	0	3	male	35.0	0	0	8.0500	NaN	S

## loc( )

Isso seleciona dados com base nos rótulos, ou seja, nomes das colunas e linhas. Nos dados acima, por exemplo, os rótulos de linha são como A/5 21171, PC17599, 113803, e as etiquetas de coluna são como PassengerId, Survived, Sex. A sintaxe geral para loc:

```
dataframe_name.loc[row_labels, column_labels]
```

### *Selecionando uma única linha.*

Insira o rótulo da linha que você deseja, ou seja, se quisermos selecionar 'Ticket' onde o valor é 'A/5 21171'.

```
# notice we need to use [] brackets

# This returns the data for the row which matches the name.
titanic_ticket_index.loc['A/5 21171']
```

```

PassengerId      1
Survived         0
Pclass          3
Sex             male
Age            22
SibSp           1
Parch           0
Fare           7.25
Cabin           NaN
Embarked        S
Name: A/5 21171, dtype: object

```

### Selecionando várias linhas

Muitas vezes precisamos selecionar várias linhas que gostaríamos de analisar mais adiante. A API `.loc` pode pegar uma lista de `row_labels` que você gostaria de selecionar, ou seja.

```

# we can supply start_label:end_label
# here we are selecting rows from label 'PC 17599' to '373450'
titanic_ticket_index.loc['PC 17599':'373450']

```

```
titanic_ticket_index.loc[['A/5 21171', 'PC 17599', '373450']]
```

executed in 12ms, finished 15:36:35 2020-07-31

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
Ticket										
A/5 21171	1	0	3	male	22.0	1	0	7.2500	NaN	S
PC 17599	2	1	1	female	38.0	1	0	71.2833	C85	C
373450	5	0	3	male	35.0	0	0	8.0500	NaN	S

Nota- Isso não funcionará se várias linhas tiverem os mesmos rótulos.

### Selecionando coluna única

É semelhante à maneira como selecionamos as linhas, mas ao selecionar colunas precisamos dizer aos Pandas as linhas que gostaríamos de selecionar. Podemos usar `:` no lugar de `row_label` o que significa que gostaríamos de selecionar todas as linhas.

```
# selecting Embarked column for all rows.
titanic_ticket_index.loc[:, 'Embarked']
```

```
Ticket
A/5 21171      S
PC 17599       C
STON/O2. 3101282 S
113803         S
373450         S
..
211536         S
112053         S
W./C. 6607     S
111369         C
370376         Q
Name: Embarked, Length: 891, dtype: object
```

### ***Selecionando várias colunas***

Semelhante ao que fizemos para várias linhas, só precisamos dizer aos Pandas quais linhas estamos selecionando.

```
# Selecting Sex, Age, Fare, Embarked column for all rows.
titanic_ticket_index.loc[:, ['Sex', 'Age', 'Fare', 'Embarked']]
```

	Sex	Age	Fare	Embarked
Ticket				
<b>A/5 21171</b>	male	22.0	7.2500	S
<b>PC 17599</b>	female	38.0	71.2833	C
<b>STON/O2. 3101282</b>	female	26.0	7.9250	S
<b>113803</b>	female	35.0	53.1000	S
<b>373450</b>	male	35.0	8.0500	S

ou como:

```
# we can supply column start_label:end_label
# here we are selecting columns from label 'Sex' to 'Embarked'
titanic_ticket_index.loc[:, 'Sex': 'Embarked']
```

	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
Ticket							
<b>A/5 21171</b>	male	22.0	1	0	7.2500	NaN	S
<b>PC 17599</b>	female	38.0	1	0	71.2833	C85	C
<b>STON/O2. 3101282</b>	female	26.0	0	0	7.9250	NaN	S
<b>113803</b>	female	35.0	1	0	53.1000	C123	S
<b>373450</b>	male	35.0	0	0	8.0500	NaN	S

### Selecionando linhas e colunas específicas

Podemos combinar a seleção em linhas e colunas para selecionar linhas e colunas específicas:

```
titanic_ticket_index.loc[['A/5 21171', 'PC 17599', '373450'], ['Sex', 'Fare', 'Age', 'Embarked']]
```

executed in 11ms, finished 15:53:42 2020-07-31

	Sex	Fare	Age	Embarked
Ticket				
<b>A/5 21171</b>	male	7.2500	22.0	S
<b>PC 17599</b>	female	71.2833	38.0	C
<b>373450</b>	male	8.0500	35.0	S

## iloc()

Isso funciona de forma semelhante ao loc, mas seleciona as linhas e colunas com base no índice em vez dos rótulos. Ao contrário dos rótulos, os índices sempre começam de 0 a number\_of\_rows-1 para linhas e 0 a number\_of\_columns-1 para colunas.

```
# selecting specific rows and columns: example 2
# we can use start_index:end_index for both columns and rows.
# selecting 3rd to 6th row and 1st to 4th column
# end_index should be 1 greater the row or column number you want
titanic_ticket_index.iloc[3:7, 1:5]
```

	Survived	Pclass	Name	Sex
Ticket				
113803	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female
373450	0	3	Allen, Mr. William Henry	male
330877	0	3	Moran, Mr. James	male
17463	0	1	McCarthy, Mr. Timothy J	male

## 7. Valores únicos nas colunas utilizando `value_counts()`

### `value_counts()`

`Value_counts` nos dá a contagem dos valores únicos em uma coluna que é muito útil para conhecer informações como

- Valores diferentes que você tem na coluna.
- Valor mais frequente.
- a proporção do valor mais frequente.

```
# value counts for the Sex column.
titanic_data['Sex'].value_counts()
# output
male      577
female    314
Name: Sex, dtype: int64
```

Como você pode ver que nosso conjunto de dados contém mais número de machos. Podemos até normalizá-lo para ver distribuição entre valores.

### `value_counts(normalize=True)`

```
# value counts normalized for Sex column
titanic_data['Sex'].value_counts(normalize=True)
#output
male      0.647587
female    0.352413
Name: Sex, dtype: float64
```

Isso significa que a proporção para o sexo masculino: feminino em nosso conjunto de dados é de aproximadamente 65:35.

## 8. Filtrar dados usando consulta( )

Normalmente, trabalhamos com grandes conjuntos de dados que são difíceis de analisar. A estratégia, nesse caso, é filtrar os dados em diferentes condições e analisá-los. Podemos fazê-lo com apenas uma linha de código usando a API de Consulta pandas.

Vamos dar alguns exemplos para entendê-lo melhor.

### *Selecione linhas com Age > 15.*

```
# first 5 records that has Age > 15
titanic_data.query('Age > 15').head(5)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

### *Selecione machos que sobreviveram.*

```
# first 5 males who survived
titanic_data.query('Sex=="male" and Survived==1').head(5)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.0000	NaN	S
21	22	1	2	Beesley, Mr. Lawrence	male	34.0	0	0	248698	13.0000	D56	S
23	24	1	1	Sloper, Mr. William Thompson	male	28.0	0	0	113788	35.5000	A6	S
36	37	1	3	Mamee, Mr. Hanna	male	NaN	0	0	2677	7.2292	NaN	C
55	56	1	1	Woolner, Mr. Hugh	male	NaN	0	0	19947	35.5000	C52	S

Podemos definir variáveis e usá-las para escrever uma consulta de filtro. Seria útil quando precisamos escrever roteiros.

```
# gender to select and min_fare, these can be passed as part of argument to the Script
gender_to_select = "female"
min_fare = 50
# querying using attribute passed
titanic_data.query('(Sex==@gender_to_select) and (Fare > @min_fare)')
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
31	32	1	Spencer, Mrs. William Augustus (Marie Eugenie)	female	NaN	1	0	PC 17569	146.5208	B78	C
52	53	1	Harper, Mrs. Henry Sleeper (Myna Haxtun)	female	49.0	1	0	PC 17572	76.7292	D33	C
61	62	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0000	B28	NaN

## 9. Salvar dados usando to\_csv ou to\_excel

Uma vez que temos os dados necessários após a seleção ou filtragem, precisamos salvá-lo para

- Compartilhe com os outros.
- Use-o em outro caderno.
- Visualize-o usando algumas ferramentas.

Como ler dados, o Pandas fornece diferentes opções para salvar os dados. Vou passar por duas APIs principais que são comumente usadas. Você pode consultar a [documentação do Pandas](#) para outros formatos nos quais você pode salvar os dados.

### to\_csv()

Isso permite que você economize em um arquivo CSV.

```
# Save data for males who survived into CSV file
males_survived = titanic_data.query('Sex=="male" and Survived==1')

# Saving to CSV, index=False i.e. do not write Index.
males_survived.to_csv('males_survived.csv', index=False)
```

### to\_excel()



Isso permite que você economize em um arquivo Excel. Pandas exigem que um pacote **Openpyxl** seja instalado. Se você não tiver, instale-o usando –

```
# If you are using Anaconda Distribution (recommended)
conda install -c conda-forge openpyxl
# install openpyxl using pip
pip install openpyxl
```

Vamos salvar em um arquivo Excel:

```
# Save the data for passengers travelling with a Sibling or Parent
passengers_not_alone = titanic_data.query('SibSp==1 | Parch==1')
# Saving to excel, index=False i.e. do not write Index.
passengers_not_alone.to_excel('travelling_with_parent_sibling.xlsx', index=False)
```

## 10. Encadeamento de múltiplas operações

Antes de terminar, vamos aprender sobre uma das características poderosas dos Pandas, ou seja, o encadeamento. Podemos aplicar várias operações no DataFrame sem salvá-lo em um dataframe temporário (*#NamesAreHard*) o que torna seu código limpo e fácil de ler.

```
# Here we first queried the data and then selected top 5 rows.
titanic_data.query('Sex=="male" and Survived==1').head(5)
```

O código acima é útil nos cenários onde não temos certeza sobre a saída. Ele ajuda a aplicar a operação e verificar diretamente as linhas superiores.

Na última seção, tivemos que primeiro criar um DataFrame temporário (*males\_survived*) para armazenar os dados dos machos que sobreviveram e depois os usamos para salvar em um arquivo CSV.

Usando o encadeamento, podemos fazer isso em apenas uma linha e até não precisamos fazer uma cópia:

```
# we used chaining to first filter and then save the records
titanic_data.query('Sex=="male" and Survived==1').to_csv('males_survived.csv', index=False)
```


Podemos praticamente acorrentar qualquer operação em Pandas desde que o resultado da última operação seja uma série ou um dataframe.

## Pandas Pt.02

Vou usar dados covid-19 de código aberto fornecidos pelo Our World In Data.

### Coronavirus Source Data

Our complete COVID-19 dataset is a collection of the COVID-19 data maintained by Our World in Data. It is updated daily and includes data on confirmed cases, deaths, and testing. All our

 <https://ourworldindata.org/coronavirus-source-data>

**Coronavirus Disease [COVID-19]**  
Research and data

by Our World in Data

Neste artigo, estaremos cobrindo com exemplos:

1. Agregação de dados usando `groupby()` e `agg()`.
2. Plotagem dados usando `plot()`.
3. Estilizar o DataFrame do Pandas usando a propriedade `.style`.
4. Pivotando os dados, convertendo dados de formato longo em formato amplo.
5. Encontrar linha e coluna com valor mínimo ou máximo usando `idxmin()` e `idxmax()`.
6. MultiIndex- simplificando suas consultas.
7. Combinando o MultiIndex em um único índice.

Vamos analisar o conjunto de dados em detalhes antes de aplicar as operações.

## Detalhes do conjunto de dados

***Our world in data*** mantém dados covid-19 que atualizam diariamente. Inclui casos confirmados, número de óbitos e testes relacionados ao Covid-19. Vamos usar dados de 7 de agosto de 2020 e ele contém 35 colunas com várias informações como PIB do país, instalações de lavagem de mãos, etc.

Para fins de demonstração, usaremos um subconjunto de dados contendo:

- `iso_code` – código alfa 3 para o país.
- `continente` - continentes do mundo.
- `localização` – país.
- `data` – data para os casos relatados.
- `new_cases` – novos casos notificados na data.
- `new_deaths` – novas mortes relatadas na data.
- `new_tests` – novos testes de corona realizados na data.

Além disso, vamos usar os dados do último dia 10 apenas de 29 de julho a 7 de agosto de 2020.

## 1. Agregação de Dados utilizando `groupby` e `agg()`

Pandas fornecem diferentes APIs para agregar os dados. Vamos ver como podemos usá-los para executar agregações simples e complexas em uma linha.

### `groupby()`

Suponha que precisamos encontrar o número total de novos casos relatados a cada dia. Podemos fazer isso usando a agregação simples `groupby()`:

```
# here we are chaining multiple operations together
# Step 1: grouping data by date.
# Step 2: selecting new_cases from the group.
# Step 3: calculating the sum of the new_cases.
# Step 4: doing a groupby changes the index, so resetting it
# Step 5: selecting Last 5 records.
data.groupby('date').new_cases.sum().reset_index().tail(5)
```

	date	new_cases
5	2020-08-03	222905.0
6	2020-08-04	204779.0
7	2020-08-05	253659.0
8	2020-08-06	274849.0
9	2020-08-07	284097.0

## Agregando múltiplos campos

No exemplo acima, nós apenas agregamos um campo, ou seja, 'new\_cases'. E se precisarmos de múltiplas agregações para um grupo? Podemos combinar `groupby()` com `agg()` nesse caso. Vamos dizer, precisamos encontrar -

- o número total de casos notificados a cada dia.
- o número máximo de casos notificados por um país para cada dia.
- o número total de óbitos notificados a cada dia.
- o número máximo de mortes relatadas por um país.
- o número total de testes realizados a cada dia
- o número total de países relatando dados por um dia.

Tudo isso pode ser feito com apenas uma linha de código, resultando em um índice **MultiColumn**:

```
# we are finding totals using sum and maximum using max
# Also, we used nunique to find unique number of countries reporting

data.groupby('date').agg({'new_cases':['sum','max'],
                          'new_deaths':['sum','max'],
                          'new_tests':['sum'],
                          'location':'nunique',
                          }).reset_index().tail(5)
```

	date	new_cases		new_deaths		new_tests	location
		sum	max	sum	max	sum	nunique
5	2020-08-03	222905.0	52972.0	3971.0	771.0	2116961.0	210
6	2020-08-04	204779.0	52050.0	4498.0	803.0	2479411.0	210
7	2020-08-05	253659.0	57525.0	6811.0	1403.0	2334686.0	209
8	2020-08-06	274849.0	57152.0	7172.0	1450.0	2214333.0	209
9	2020-08-07	284097.0	62538.0	6910.0	1848.0	811897.0	209

## Nomeando agregações

Se você notou a saída acima, ela nos deu os resultados, mas os nomes das colunas não são amigáveis, pois está apenas usando nomes de coluna existentes. Podemos usar pandas para nomear cada agregação na saída.

Para o mesmo problema acima, podemos fazer:

```
# naming the aggregations, you can use any name for the aggregate
filtered_data.groupby('date').agg(
    total_new_cases = ('new_cases', 'sum'),
    max_new_cases_country = ('new_cases', 'max'),
    total_new_deaths = ('new_deaths', 'sum'),
    max_new_deaths_country = ('new_deaths', 'max'),
    total_new_tests = ('new_tests', 'sum'),
    total_countries_reported = ('location', 'nunique')
).reset_index().tail(5)
```

	date	total_new_cases	max_new_cases_country	total_new_deaths	max_new_deaths_country	total_new_tests	total_countries_reported
5	2020-08-03	222905.0	52972.0	3971.0	771.0	2116961.0	210
6	2020-08-04	204779.0	52050.0	4498.0	803.0	2479411.0	210
7	2020-08-05	253659.0	57525.0	6811.0	1403.0	2334686.0	209
8	2020-08-06	274849.0	57152.0	7172.0	1450.0	2214333.0	209
9	2020-08-07	284097.0	62538.0	6910.0	1848.0	811897.0	209

## 2. Plotagem de dados usando Plotly( )

O principal trabalho de um analista de dados é visualizar os dados. Cada DataFrame em Pandas tem uma API de plot () para plotagem.

Por padrão, Pandas usa Matplotlib como um backend para a trama. Existem vários outros backends que você pode usar.

Aqui, usaremos a biblioteca Plotly e configurá-la como os pandas plotando backend. Deveremos instalar a biblioteca plotly:

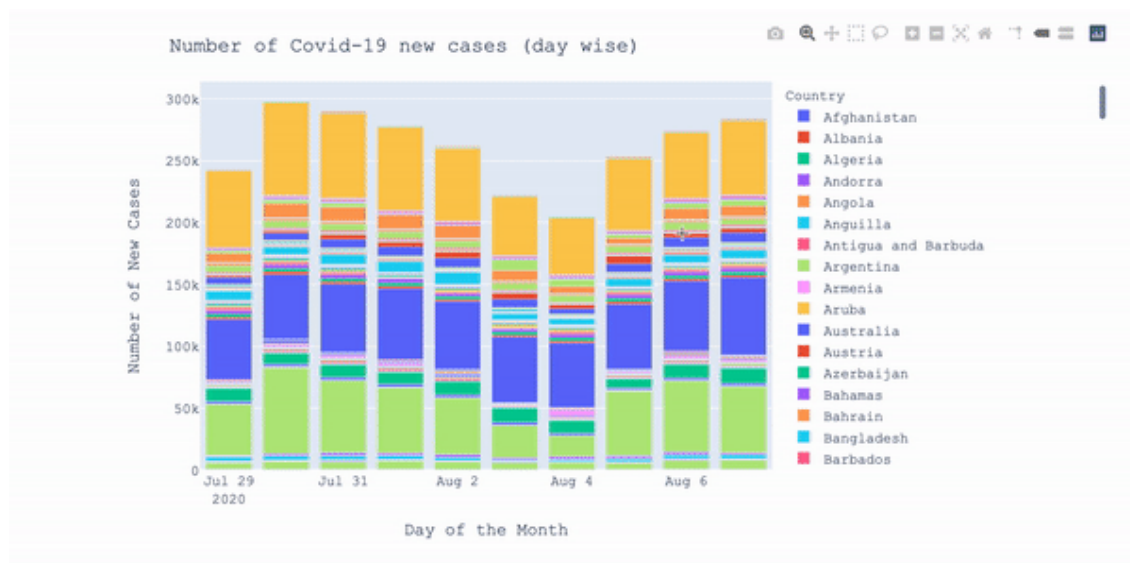
```
# if using anaconda distribution (recommended)
conda install -c conda-forge plotly

# if using pip
pip install plotly
```

## Novos Casos relatados por cada país para cada dia.

Digamos que precisamos traçar novos casos relatados por cada país para cada dia. Podemos fazer isso usando:

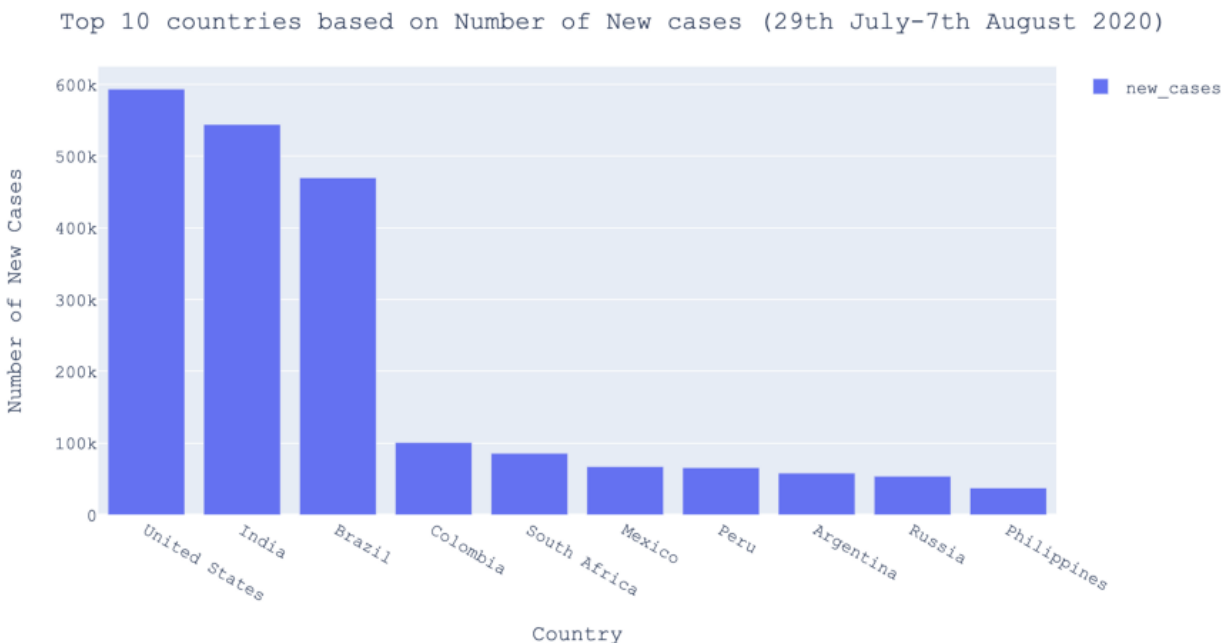
```
# generating a bar plot
data.plot.bar(x='date', y='new_cases', color='location')
```



## Top 10 países com base no número de novos casos nos últimos 10 dias

Para obter os principais países, primeiro precisamos agrupar os dados por 'localização' e depois plotá-lo usando:

```
# Step 1: generating a new dataset based on each location i.e. country
# Step 2: doing the sum on new_cases followed by sorting
# Step 3: Selecting top 10 countries from the dataset
data.groupby(['location']).new_cases.sum().sort_values(ascending=False).head(10).plot.bar()
```



### 3. Estilo de dados usando a API de estilo

Muitas vezes precisamos alternar entre Excel e Pandas para fazer coisas diferentes, como formatar os dados ou adicionar alguns estilos. Pandas introduziram API **styling** que pode ser usado para estilizar o conjunto de dados em Pandas em si. Podemos fazer várias operações usando estilo. Vamos passar por alguns exemplos.

#### Converter números em separados por vírgulas.

Em nosso conjunto de dados, `new_cases`, `new_deaths` e `new_tests` são armazenados como um valor flutuante que não se encaixa para apresentação. Usando estilos podemos alterá-los para valores separados por vírgulas como

```
# formatting the data to show numbers as comma separated
data.style.format({'new_cases': '{0:,.0f}'},
```

```
'new_deaths': '{0:,.0f}',
'new_tests': '{0:,.0f}',}).hide_index()
```

iso_code	continent	location	date	new_cases	new_deaths	new_tests
USA	North America	United States	2020-07-30	74,985	1,457	805,854
BRA	South America	Brazil	2020-07-30	69,074	1,595	nan
USA	North America	United States	2020-07-31	68,032	1,357	725,982
USA	North America	United States	2020-08-01	67,023	1,244	726,010
IND	Asia	India	2020-08-07	62,538	886	639,042

Podemos usá-lo para fazer outras coisas extravagantes, como adicionar um sinal de moeda para o campo de quantidade, alterar o número de pontos decimais, etc.

## Destacando os valores máximos.


Muitas vezes, precisamos destacar o valor máximo na coluna. Pode ser feito usando:

```
# This will highlight the maximum for numeric column
data.style.highlight_max().hide_index()
```

## Formatação condicional com Pandas:

Styling - pandas 1.2.0 documentation

This document is written as a Jupyter Notebook, and can be viewed or downloaded here. You can apply conditional formatting, the visual styling of a DataFrame depending on the data within, by using the DataFrame.style property. This is a property that returns a object, which has useful methods for

 [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/style.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/style.html)

## Mapa de cores baseado na magnitude dos valores.

### background\_gradient()

Às vezes, é útil ver um mapa de cores baseado na magnitude dos valores, ou seja, grandes valores exibidos com cor escura e pequenos valores com luz. Pandas Styling fornece uma API agradável que faz isso perfeitamente com uma linha de código:



```
# Adding blue color map for each numeric field
data.style.hide_index().background_gradient(cmap='Blues')
```

## 4. Pivô- converter dados de formato longo para amplo

Se você está familiarizado com o Excel, você teria ouvido falar de tabelas pivôs. O pivô pandas nos ajuda a converter dados longos, ou seja, armazenados em linhas para dados amplos, ou seja, em colunas. Pandas fornecem duas APIs diferentes para pivotar:

- Pivô
- mesa pivô

Vamos passar por cima de cada um usando alguns exemplos.

### pivô( )

Considere um problema que precisamos encontrar covid-19 novos casos mudando horas extras para os 10 principais países.

Primeiro, precisamos encontrar os 10 principais países para cada dia que podemos fazer usando abaixo do código

```
# Step 1: create a group based on date and location

# Step 2: order it by number of new cases.
grouped_data = data.groupby(['date', 'location']).new_cases.sum().sort_values(ascending=False)
# we have data for each date grouped by location.
grouped_data.head(5)
```

Output:

date	location	
2020-07-30	United States	74985.0
	Brazil	69074.0
2020-07-31	United States	68032.0
2020-08-01	United States	67023.0
2020-08-07	India	62538.0

```
# Next we need to select top 10 countries for each date
# Step 3: create a new group based on date.
# Step 4: select top 10 records from each group.
top10_countries = grouped_data.groupby('date').head(10).reset_index()
```

	date	location	new_cases
0	2020-07-30	United States	74985.0
1	2020-07-30	Brazil	69074.0
2	2020-07-31	United States	68032.0
3	2020-08-01	United States	67023.0
4	2020-08-07	India	62538.0

Agora, temos os 10 melhores países para cada dia, mas isso é armazenado no formato longo, ou seja, para cada dia temos várias linhas. Podemos usar o pivô para converter esses dados em amplo formato usando:

```
# Step 5: pivoting data on date and location
top10_countries_pivot = top10_countries.pivot(index='date',
columns='location',
values='new_cases')
```

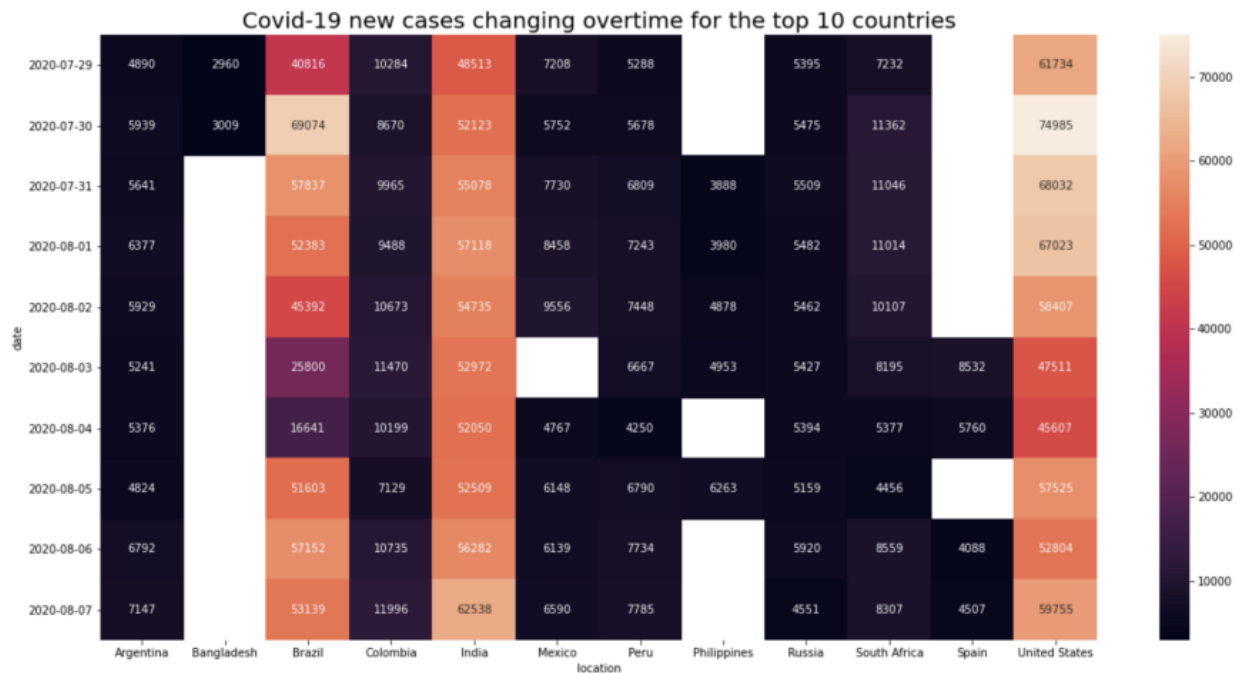
No comando acima, estamos dizendo que para cada data, criar novas colunas para cada local com os valores que temos para new\_cases. Isso resultará em:

location	Argentina	Bangladesh	Brazil	Colombia	India	Mexico	Peru	Philippines	Russia	South Africa	Spain	United States
date												
2020-08-03	5241.0	NaN	25800.0	11470.0	52972.0	NaN	6667.0	4953.0	5427.0	8195.0	8532.0	47511.0
2020-08-04	5376.0	NaN	16641.0	10199.0	52050.0	4767.0	4250.0	NaN	5394.0	5377.0	5760.0	45607.0
2020-08-05	4824.0	NaN	51603.0	7129.0	52509.0	6148.0	6790.0	6263.0	5159.0	4456.0	NaN	57525.0
2020-08-06	6792.0	NaN	57152.0	10735.0	56282.0	6139.0	7734.0	NaN	5920.0	8559.0	4088.0	52804.0
2020-08-07	7147.0	NaN	53139.0	11996.0	62538.0	6590.0	7785.0	NaN	4551.0	8307.0	4507.0	59755.0

## Seaborn

Em seguida, podemos gerar um bom mapa de calor para os dados pivotados diretamente usando o Seaborn:

```
# Step 6: plotting heatmap using Seaborn
sns.heatmap(top10_countries_pivot, annot=True, fmt='.0f')
```



## **pivot\_table.**

No caso acima, tínhamos apenas uma linha para a combinação de (data, localização) ou seja, para 2020-08-07, tínhamos apenas uma linha para 'Índia'.

**E se precisarmos encontrar o número de novos casos para cada continente?**

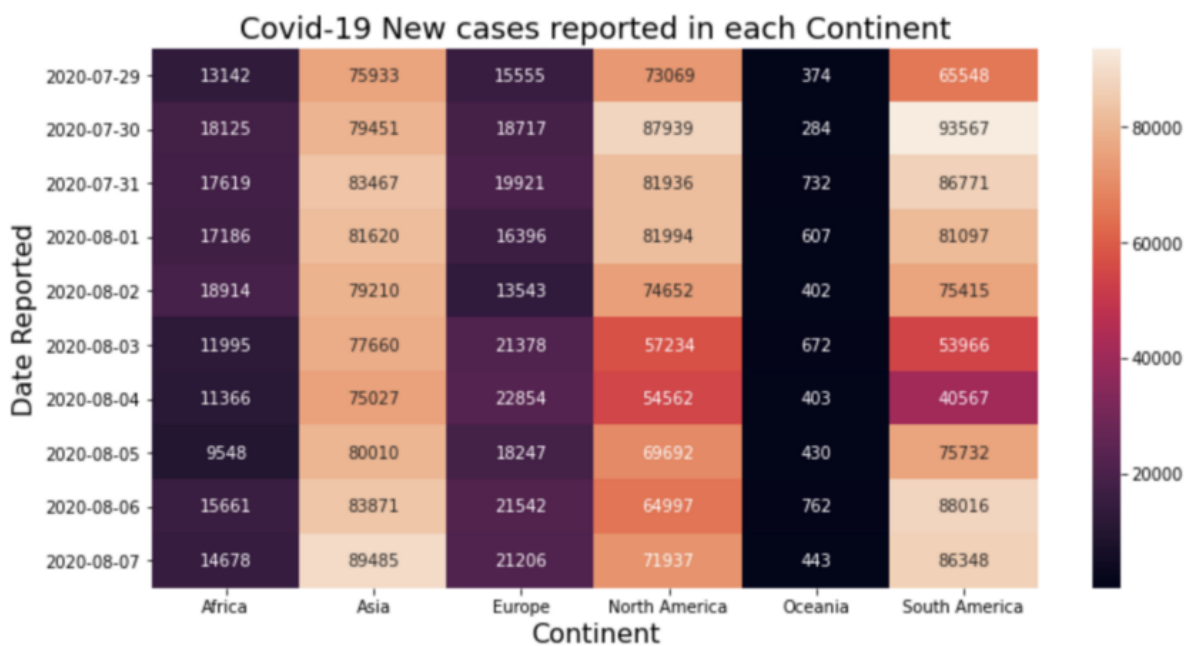
Neste caso, teríamos várias linhas para cada continente em uma determinada data, ou seja, o número de linhas seria igual ao número de países do continente.

Para superar esse problema, a Pandas fornece **pivot\_table()** API que pode **agregar registros para convertê-lo em uma única linha**. Vamos ver como isso pode ser feito:

## **Criando pivot por continente**

```
# Creating a pivot table for continent
# We do not need to select top 10 records here as we have only 6 continents
# Notice the aggfunc below, it will actually sum the new_cases for each country in the continent.
continent_pivot = filtered_data.pivot_table(index='date',
columns='continent', values='new_cases', aggfunc='sum')
```

continent	Africa	Asia	Europe	North America	Oceania	South America
date						
2020-07-29	13142.0	75933.0	15555.0	73069.0	374.0	65548.0
2020-07-30	18125.0	79451.0	18717.0	87939.0	284.0	93567.0
2020-07-31	17619.0	83467.0	19921.0	81936.0	732.0	86771.0
2020-08-01	17186.0	81620.0	16396.0	81994.0	607.0	81097.0
2020-08-02	18914.0	79210.0	13543.0	74652.0	402.0	75415.0
2020-08-03	11995.0	77660.0	21378.0	57234.0	672.0	53966.0
2020-08-04	11366.0	75027.0	22854.0	54562.0	403.0	40567.0
2020-08-05	9548.0	80010.0	18247.0	69692.0	430.0	75732.0
2020-08-06	15661.0	83871.0	21542.0	64997.0	762.0	88016.0
2020-08-07	14678.0	89485.0	21206.0	71937.0	443.0	86348.0



## 5. Utilizando `idxmin( )`, `idxmax( )`

Como você acha que o país tem casos mínimos ou máximos na tabela de pivôs? Podemos fazê-lo usando `idxmin()` ou `idxmax()`. O `Idxmin()` nos dá o índice do valor mínimo para um determinado eixo, ou seja, linhas ou colunas e `idxmax()` nos dá o índice do valor máximo.

```
# find country with maximum cases
# axis=1 to find max in the row, default is column
top10_countries_pivot.idxmax(axis=1)
Output:
date
2020-07-29    United States
2020-07-30    United States
2020-07-31    United States
2020-08-01    United States
2020-08-02    United States
2020-08-03             India
2020-08-04             India
2020-08-05    United States
2020-08-06             Brazil
2020-08-07             India
```

```
# find country with minimum cases
# axis=1 to find max in the row, default is column
top10_countries_pivot.idxmin(axis=1)
# This is among the top 10 countries
Output:
date
2020-07-29    Bangladesh
2020-07-30    Bangladesh
2020-07-31    Philippines
2020-08-01    Philippines
2020-08-02    Philippines
2020-08-03    Philippines
2020-08-04             Peru
2020-08-05    South Africa
2020-08-06             Spain
2020-08-07             Spain
```

## 6. MultiIndex usando `set_index ( )`

Pandas suportam multi-indexação para linhas e colunas. Isso é útil para responder a perguntas simples. Por exemplo, se precisarmos encontrar dados —

- para um dia específico para um continente específico.
- para um local específico em um dia específico.
- alguns locais em alguns dias.

Existem muitas maneiras de fazer isso, mas vamos ver como é fácil usar multiíndice. Primeiro, criar vários índices usando a API `set_index`:

```
# Creating Index on continent, location and date
# The index will be created in the order supplied
indexed_data = data.set_index(['continent', 'location', 'date']).sort_index()
# values of the index
indexed_data.index.values
Output:
array([('Africa', 'Algeria', '2020-07-29'),
      ('Africa', 'Algeria', '2020-07-30'),
      ('Africa', 'Algeria', '2020-07-31'), ...,
      ('South America', 'Venezuela', '2020-08-05'),
      ('South America', 'Venezuela', '2020-08-06'),
      ('South America', 'Venezuela', '2020-08-07')], dtype=object)
```

Agora, temos vários índices para cada linha, ou seja, para consultar a primeira linha, precisamos fazer:

```
indexed_data.loc[('Africa', 'Algeria', '2020-07-29')]
```

			iso_code	new_cases	new_deaths	new_tests
continent	location	date				
Africa	Algeria	2020-07-29	DZA	642.0	11.0	NaN
		2020-07-30	DZA	614.0	12.0	NaN
		2020-07-31	DZA	602.0	14.0	NaN
		2020-08-01	DZA	563.0	10.0	NaN
		2020-08-02	DZA	556.0	13.0	NaN
...	...	...	...	...	...	...
South America	Venezuela	2020-08-03	VEN	763.0	5.0	NaN
		2020-08-04	VEN	0.0	0.0	NaN
		2020-08-05	VEN	1232.0	13.0	NaN
		2020-08-06	VEN	861.0	8.0	NaN
		2020-08-07	VEN	981.0	7.0	NaN

Vamos ver alguns exemplos de como criar um multiíndice é útil.

## 0 número de novos casos notificados nos EUA (Localização) da América do Norte (Continente) em 7 de agosto de 2020.

```
indexed_data.loc(['North America','United States','2020-08-07'],  
                 'new_cases']
```

Output:  
59755.0

## Novos casos notificados na Ásia em 7 de agosto de 2020

Aqui gostaríamos de obter dados para todos os países da Ásia. Podemos fazê-lo passando `fatia` (Nenhum) no local de localização que significa obter todos os locais.

```
indexed_data.loc(['Asia',slice(None),'2020-08-07'],'new_cases']
```

Output(a snippet):

continent	location	date	
Asia	Afghanistan	2020-08-07	41.0
	Armenia	2020-08-07	233.0
	Azerbaijan	2020-08-07	144.0
	Bahrain	2020-08-07	375.0
	Bangladesh	2020-08-07	2977.0
	Bhutan	2020-08-07	3.0
	Brunei	2020-08-07	0.0
	Cambodia	2020-08-07	0.0
	China	2020-08-07	132.0
	Georgia	2020-08-07	0.0
	India	2020-08-07	62538.0

## Novos casos notificados na Índia e nos Estados Unidos nos dias 6 e 7 de agosto

Podemos passar uma lista de valores para qualquer índice. Neste exemplo, não estamos fornecendo o continente e selecionando vários valores para localização e data.

```
indexed_data.loc[(slice(None),['India','United States'],['2020-08-06','2020-08-07']),  
                 'new_cases']Output:  
continent    location    date  
Asia         India      2020-08-06    56282.0
```

```


                2020-08-07    62538.0
North America  United States  2020-08-06    52804.0
                2020-08-07    59755.0
Name: new_cases, dtype: float64

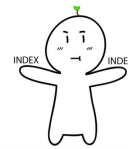
```

Podemos fazer muitas coisas com multiíndice. Para aprender mais do que se referir a um bom artigo escrito por Byron Dolon 'How to Use MultiIndex in Pandas to Level Up Your Analysis':

#### How to Use MultiIndex in Pandas to Level Up Your Analysis

What if you could have more than one column as in your DataFrame's index? The multi-level index feature in Pandas allows you to do just that. A regular Pandas DataFrame has a single

 <https://towardsdatascience.com/how-to-use-multiindex-in-pandas-to-level-up-your-analysis-aeac7f451fce>



## 7. Combinando o MultiIndex em um único índice.

Muitas vezes, quando fazemos agregação, temos índices MultiColumn como:

```

# multiple aggregations on new_cases
grouped_data = data.groupby('date').agg({'new_cases': ['sum', 'max', 'min']})

```



date	new_cases		
	sum	max	min
<b>2020-07-29</b>	243621.0	61734.0	0.0
<b>2020-07-30</b>	298083.0	74985.0	0.0
<b>2020-07-31</b>	290446.0	68032.0	0.0
<b>2020-08-01</b>	278900.0	67023.0	0.0
<b>2020-08-02</b>	262136.0	58407.0	0.0

Isso torna um pouco difícil obter os resultados reais. Podemos convertê-los em uma única coluna seguindo:

```
# columns in grouped data
grouped_data.columns
Output:
MultiIndex([('new_cases', 'sum'),
            ('new_cases', 'max'),
            ('new_cases', 'min')],
           )
```

Combinar as colunas acima como new\_cases\_sum, new\_cases\_max new\_cases\_min usando o código simples:

```
# here are we just joining the tuple with '_'
# this works for level-2 column indexes only
new_columns = ['%s%s' % (a, '_%s' % b if b else '') for a, b in grouped_data.columns]
new_columns
Output:
['new_cases_sum', 'new_cases_max', 'new_cases_min']
# change grouped_data columns.
grouped_data.columns = new_columns
```

	<b>date</b>	<b>new_cases_sum</b>	<b>new_cases_max</b>	<b>new_cases_min</b>
<b>0</b>	2020-07-29	243621.0	61734.0	0.0
<b>1</b>	2020-07-30	298083.0	74985.0	0.0
<b>2</b>	2020-07-31	290446.0	68032.0	0.0
<b>3</b>	2020-08-01	278900.0	67023.0	0.0
<b>4</b>	2020-08-02	262136.0	58407.0	0.0
<b>5</b>	2020-08-03	222905.0	52972.0	0.0
<b>6</b>	2020-08-04	204779.0	52050.0	0.0
<b>7</b>	2020-08-05	253659.0	57525.0	0.0
<b>8</b>	2020-08-06	274849.0	57152.0	0.0
<b>9</b>	2020-08-07	284097.0	62538.0	0.0