**Para saber mais: aprofundando na normalização**

**Normalização** de dados é um processo importante em ciência de dados que tem como objetivo organizar e padronizar dados para facilitar a análise e comparação entre eles. Quando se trata de dados no formato JSON (*JavaScript Object Notation* - Notação de Objetos JavaScript), é comum que eles estejam aninhados, o que pode dificultar sua análise e manipulação.

A biblioteca Pandas possui uma função chamada json\_normalize() que permite a transformação dos dados em um formato tabular, facilitando a visualização e análise das informações. A seguir vamos aprender como usar essa função para normalizar diferentes tipos de JSON em DataFrames.

**Normalizando um JSON simples**

Abaixo temos uma variável chamada dados e dentro dela há **um objeto** JSON com três chaves e seus respectivos valores:

dados = {'Pesquisa': 'Principais Indicadores de Doenca Cardiaca', 'Ano': 2020, 'Numero\_Pacientes':3}

Copiar código

Para normalizar essa variável, podemos passa-lá dentro da função json\_normalize e analisar o DataFrame obtido:

df = pd.json\_normalize(dados)

df

| **index** | **Pesquisa** | **Ano** | **Numero\_Pacientes** |
| --- | --- | --- | --- |
| 0 | Principais Indicadores de Doenca Cardiaca | 2020 | 3 |

O resultado parece ótimo! Temos 3 colunas que são as nossas chaves e uma linha contendo os valores para cada chave.

**Normalizando um JSON com vários níveis**

Existem situações em que o arquivo JSON pode conter **mais de um objeto**, como é o caso do exemplo abaixo, no qual temos uma **lista**, armazenada na variável json\_lista, contendo **dois objetos JSON**:

json\_lista = [

{ 'ID': '01', 'Faixa\_etaria': '55-59', 'Sexo\_biologico': 'feminino'},

{ 'ID': '02', 'Faixa\_etaria': '80 ou +', 'Sexo\_biologico': 'feminino'}

]

Copiar código

Para normalizar essa lista podemos aplicar a função json\_normalize:

pd.json\_normalize(json\_lista)

| **index** | **ID** | **Faixa\_etaria** | **Sexo\_biologico** |
| --- | --- | --- | --- |
| 0 | 01 | 55-59 | feminino |
| 1 | 02 | 80 ou + | feminino |

A função json\_normalize() é capaz de converter cada registro da lista em uma linha de forma tabular.

**Normalizando um JSON com uma lista aninhada**

Bom, notamos que a função json\_normalize() funciona muito bem nas situações anteriores, mas e em outras situações?

**Dados como um dicionário**

Vamos começar analisando a normalização quando os dados são um dicionário. Temos um dicionário armazenado na variável json\_obj. Note que na chave “Saude” temos outro dicionário:

json\_obj = {

'ID': '01',

'Faixa\_etaria': '55-59',

'Sexo\_biologico': 'Feminino',

'Saude': {'Dificuldade\_caminhar': 'Nao',

'Atividade\_fisica': 'Sim',

'IMC': 16.6,

'Doenca\_cardiaca': 'Nao',

}

}

Fazendo a normalização:

pd.json\_normalize(json\_obj)

| **index** | **ID** | **Faixa\_etaria** | **Sexo\_biologico** | **Saude.Dificuldade\_caminhar** | **Saude.Atividade\_fisica** | **Saude.IMC** | **Saude.Doenca\_cardiaca** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 01 | 55-59 | Feminino | Nao | Sim | 16.6 | Nao |

O DataFrame gerado possui uma coluna para cada informação contida no dicionário que estava em “Saude”. As colunas criadas possuem o prefixo “Saude.”, pois as informações vieram dessa mesma chave.

**Dados como uma lista de dicionários**

Agora nós temos uma lista de dicionários armazenada na variável json\_list:

json\_list = [

{

'ID': '01',

'Faixa\_etaria': '55-59',

'Sexo\_biologico': 'Feminino',

'Saude': {'Dificuldade\_caminhar': 'Nao',

'Atividade\_fisica': 'Sim',

'IMC': 16.6,

'Doenca\_cardiaca': 'Nao',

}

},

{

'ID': '02',

'Faixa\_etaria': '80 ou +',

'Sexo\_biologico': 'Feminino',

'Saude': {'Dificuldade\_caminhar': 'Nao',

'Atividade\_fisica': 'Sim',

'IMC': 20.34,

'Doenca\_cardiaca': 'Sim'}

}

]

Fazendo a normalização:

pd.json\_normalize(json\_list)

| **index** | **ID** | **Faixa\_etaria** | **Sexo\_biologico** | **Saude.Dificuldade\_caminhar** | **Saude.Atividade\_fisica** | **Saude.IMC** | **Saude.Doenca\_cardiaca** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 01 | 55-59 | Feminino | Nao | Sim | 16.6 | Nao |
| 1 | 02 | 80 ou + | Feminino | Nao | Sim | 20.34 | Sim |

Podemos analisar que todos os valores aninhados em cada registro da lista foram convertidos em colunas separadas. E os dados que nós normalizamos no vídeo anterior? Lembra que a normalização deles foi feita de um jeito diferente?

Recapitulando, vamos copiar os dados do arquivo [pacientes\_2.json](https://github.com/alura-cursos/Pandas/blob/main/pacientes_2.json) e armazená-los numa variável chamada dados\_dict.

dados\_dict = {

"Pesquisa": "Principais Indicadores de Doenca Cardiaca",

"Ano": 2020,

"Pacientes": [

{

"ID": "01",

"Faixa\_etaria": "55-59",

"Sexo\_biologico": "Feminino",

"Raça": "Branca",

"IMC": 16.6,

"Fumante": "Sim",

"Consumo\_alcool": "Nao",

"Saude\_fisica": 3,

"Saude\_mental": 30,

"Dificuldade\_caminhar": "Nao",

"Atividade\_fisica": "Sim",

"Saude\_geral": "Muito boa",

"Horas\_sono": 5,

"Problemas\_saude": [

"Diabetes",

"Asma",

"Cancer\_pele"

]

},

{

"ID": "02",

"Faixa\_etaria": "80 ou +",

"Sexo\_biologico": "Feminino",

"Raça": "Branca",

"IMC": 20.34,

"Fumante": "Nao",

"Consumo\_alcool": "Nao",

"Saude\_fisica": 0,

"Saude\_mental": 0,

"Dificuldade\_caminhar": "Nao",

"Atividade\_fisica": "Sim",

"Saude\_geral": "Muito boa",

"Horas\_sono": 7,

"Problemas\_saude": [

"AVC"

]

},

{

"ID": "03",

"Faixa\_etaria": "65-69",

"Sexo\_biologico": "Masculino",

"Raça": "Branca",

"IMC": 26.58,

"Fumante": "Sim",

"Consumo\_alcool": "Nao",

"Saude\_fisica": 20,

"Saude\_mental": 30,

"Dificuldade\_caminhar": "Nao",

"Atividade\_fisica": "Sim",

"Saude\_geral": "Muito boa",

"Horas\_sono": 8,

"Problemas\_saude": [

"diabetes",

"Asma"

]

}

]

}

Se tentarmos normalizar esses dados:

pd.json\_normalize(dados\_dict)

| **index** | **Pesquisa** | **Ano** | **Pacientes** |
| --- | --- | --- | --- |
| 0 | Principais Indicadores de Doenca Cardiaca | 2020 | {'ID': '01', 'Faixa\_etaria': '55-59', 'Sexo\_b... |

Podemos observar que nossa lista aninhada é colocada em uma única coluna *Pacientes*. Então, nós usamos o seguinte código para normalizar os dados, especificando qual coluna está aninhada:

pd.json\_normalize(dados\_dict['Pacientes'])

Também podemos fazer isso utilizando o parâmetro record\_path como ['Pacientes']. Esse parâmetro é usado na função pd.json\_normalize() para especificar o caminho para os registros que devem ser normalizados em um DataFrame separado:

pd.json\_normalize(dados\_dict, record\_path=['Pacientes'])

| **index** | **ID** | **Faixa\_etaria** | **Sexo\_biologico** | **Raça** | **...** | **Atividade\_fisica** | **Saude\_geral** | **Horas\_sono** | **Problemas\_saude** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 01 | 55-59 | Feminino | Branca | ... | Sim | Muito boa | 5 | Diabetes,Asma,Cancer\_pele |
| 1 | 02 | 80 ou + | Feminino | Branca | ... | Sim | Muito boa | 7 | AVC |
| 2 | 03 | 65-69 | Masculino | Branca | ... | Sim | Muito boa | 8 | diabetes,Asma |

Com ambos os códigos o resultado é o mesmo.

O resultado parece ótimo, mas não inclui as colunas “Pesquisa” e “Ano”. Para incluí-las, podemos usar o parâmetro meta para especificar outras colunas que queremos no DataFrame.

pd.json\_normalize(

dados\_dict,

record\_path =['Pacientes'],

meta=['Pesquisa', 'Ano']

)

Copiar código

| **index** | **ID** | **Faixa\_etaria** | **Sexo\_biologico** | **...** | **Problemas\_saude** | **Pesquisa** | **Ano** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 01 | 55-59 | Feminino | ... | [Diabetes,Asma,Cancer\_pele] | Principais Indicadores de Doenca Cardiaca | 2020 |
| 1 | 02 | 80 ou + | Feminino | ... | AVC | Principais Indicadores de Doenca Cardiaca | 2020 |
| 2 | 03 | 65-69 | Masculino | ... | [diabetes,Asma] | Principais Indicadores de Doenca Cardiaca | 2020 |

Dessa forma temos todas as colunas presentes no DataFrame!

**Importante**: Na aula nós realizamos a normalização em um arquivo com o formato JSON. No entanto, a função json\_normalize() aceita apenas um dicionário ou uma lista de dicionários. Por isso, no vídeo foi usada a estratégia de usar o código: pd.json\_normalize(dados\_pacientes\_2['Pacientes']). Porém, se tentarmos usar parâmetros da função json\_normalize em um arquivo JSON podem surgir erros. Para contornar isso, precisamos importar o **módulo json** e ler os arquivos conforme o código abaixo:

#Importando a biblioteca Pandas

**import** pandas **as** pd

#Importando o módulo JSON

**import** json

#Lendo o arquivo json usando o módulo Python JSON

**with** open('pacientes\_2.json','r') **as** f:

dados = json.loads(f.read())

#Normalizando os dados com os parâmetros record\_path e meta

pd.json\_normalize(dados, record\_path='Pacientes', meta=['Pesquisa', 'Ano'])

E, assim, aprendemos como normalizar arquivos JSON simples, com múltiplos níveis e aninhados.

**Para saber mais: obtendo arquivos JSON de APIs**

Para obter dados no formato JSON podemos utilizar uma **API** (*Application Programming Interface*). Ela é uma interface de programação de aplicação que permite a comunicação entre diferentes programas ou sistemas.

**Atenção**: Caso queira aprofundar no assunto, temos um [Alura +](https://cursos.alura.com.br/extra/alura-mais/o-que-e-uma-api--c697) explicando o que é uma API.

Existem vários formatos que podem ser usados em APIs e um dos mais comuns é o **JSON** (*JavaScript Object Notation*). O JSON é um formato leve de troca de informações entre sistemas que utiliza uma notação baseada em objetos do **JavaScript**.

Nós temos a seguinte situação-problema: “Uma empresa de tecnologia está em fase de expansão e deseja otimizar sua base de contatos e a forma como gerencia suas interações com os clientes e parceiros. A equipe de vendas e atendimento ao cliente está enfrentando dificuldades para acompanhar todos os contatos, entender as necessidades dos diferentes perfis de clientes, e personalizar o atendimento de forma eficiente. Para solucionar isso, a empresa precisa de uma maneira organizada e automatizada para coletar, gerenciar e segmentar informações de seus clientes e parceiros, como nome, e-mail, endereço, telefone e empresa em que trabalham.”

Nesta situação, podemos acessar a API [JSONPlaceholder](https://jsonplaceholder.typicode.com/users" \t "_blank) para coletar os dados dos contatos, que incluem nome, e-mail, endereço, telefone, entre outros. Com esses dados, a equipe pode melhorar suas operações e a comunicação com os clientes.

Uma maneira comum de acessar APIs é por meio da biblioteca [requests](https://pypi.org/project/requests/" \t "_blank). Para isso, também é necessário importar o módulo json:

**import** requests

**import** json

Os dados da **JSONPlaceholder** serão obtidos com a função request.get('url'). Dentro dela nós passamos o endereço da API:

dados\_usuarios = requests.get('https://jsonplaceholder.typicode.com/users')

Conseguimos recuperar os resultados usando a função json.loads(). Para isso, é necessário passar a variável dados\_usuarios, criada anteriormente com o atributo text que retorna o conteúdo da resposta.

resultado = json.loads(dados\_usuarios.text)

Em seguida, podemos visualizar o DataFrame.

pd.DataFrame(resultado)

| **index** | **id** | **name** | **username** | **email** | **address** | **phone** | **website** | **company** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | Leanne Graham | Bret | Sincere@april.biz | {'street': 'Kulas Light', 'suite': 'Apt. 556', 'city': 'Gwenborough', 'zipcode': '92998-3874', 'geo': {'lat': '-37.3159', 'lng': '81.1496'}} | 1-770-736-8031 x56442 | hildegard.org | {'name': 'Romaguera-Crona', 'catchPhrase': 'Multi-layered client-server neural-net', 'bs': 'harness real-time e-markets'} |
| 1 | 2 | Ervin Howell | Antonette | Shanna@melissa.tv | {'street': 'Victor Plains', 'suite': 'Suite 879', 'city': 'Wisokyburgh', 'zipcode': '90566-7771', 'geo': {'lat': '-43.9509', 'lng': '-34.4618'}} | 010-692-6593 x09125 | anastasia.net | {'name': 'Deckow-Crist', 'catchPhrase': 'Proactive didactic contingency', 'bs': 'synergize scalable supply-chains'} |
| 2 | 3 | Clementine Bauch | Samantha | Nathan@yesenia.net | {'street': 'Douglas Extension', 'suite': 'Suite 847', 'city': 'McKenziehaven', 'zipcode': '59590-4157', 'geo': {'lat': '-68.6102', 'lng': '-47.0653'}} | 1-463-123-4447 | ramiro.info | {'name': 'Romaguera-Jacobson', 'catchPhrase': 'Face to face bifurcated interface', 'bs': 'e-enable strategic applications'} |
| 3 | 4 | Patricia Lebsack | Karianne | Julianne.OConner@kory.org | {'street': 'Hoeger Mall', 'suite': 'Apt. 692', 'city': 'South Elvis', 'zipcode': '53919-4257', 'geo': {'lat': '29.4572', 'lng': '-164.2990'}} | 493-170-9623 x156 | kale.biz | {'name': 'Robel-Corkery', 'catchPhrase': 'Multi-tiered zero tolerance productivity', 'bs': 'transition cutting-edge web services'} |
| 4 | 5 | Chelsey Dietrich | Kamren | Lucio\_Hettinger@annie.ca | {'street': 'Skiles Walks', 'suite': 'Suite 351', 'city': 'Roscoeview', 'zipcode': '33263', 'geo': {'lat': '-31.8129', 'lng': '62.5342'}} | (254)954-1289 | demarco.info | {'name': 'Keebler LLC', 'catchPhrase': 'User-centric fault-tolerant solution', 'bs': 'revolutionize end-to-end systems'} |
| 5 | 6 | Mrs. Dennis Schulist | Leopoldo\_Corkery | Karley\_Dach@jasper.info | {'street': 'Norberto Crossing', 'suite': 'Apt. 950', 'city': 'South Christy', 'zipcode': '23505-1337', 'geo': {'lat': '-71.4197', 'lng': '71.7478'}} | 1-477-935-8478 x6430 | ola.org | {'name': 'Considine-Lockman', 'catchPhrase': 'Synchronised bottom-line interface', 'bs': 'e-enable innovative applications'} |
| 6 | 7 | Kurtis Weissnat | Elwyn.Skiles | Telly.Hoeger@billy.biz | {'street': 'Rex Trail', 'suite': 'Suite 280', 'city': 'Howemouth', 'zipcode': '58804-1099', 'geo': {'lat': '24.8918', 'lng': '21.8984'}} | 210.067.6132 | elvis.io | {'name': 'Johns Group', 'catchPhrase': 'Configurable multimedia task-force', 'bs': 'generate enterprise e-tailers'} |
| 7 | 8 | Nicholas Runolfsdottir V | Maxime\_Nienow | Sherwood@rosamond.me | {'street': 'Ellsworth Summit', 'suite': 'Suite 729', 'city': 'Aliyaview', 'zipcode': '45169', 'geo': {'lat': '-14.3990', 'lng': '-120.7677'}} | 586.493.6943 x140 | jacynthe.com | {'name': 'Abernathy Group', 'catchPhrase': 'Implemented secondary concept', 'bs': 'e-enable extensible e-tailers'} |
| 8 | 9 | Glenna Reichert | Delphine | Chaim\_McDermott@dana.io | {'street': 'Dayna Park', 'suite': 'Suite 449', 'city': 'Bartholomebury', 'zipcode': '76495-3109', 'geo': {'lat': '24.6463', 'lng': '-168.8889'}} | (775)976-6794 x41206 | conrad.com | {'name': 'Yost and Sons', 'catchPhrase': 'Switchable contextually-based project', 'bs': 'aggregate real-time technologies'} |
| 9 | 10 | Clementina DuBuque | Moriah.Stanton | Rey.Padberg@karina.biz | {'street': 'Kattie Turnpike', 'suite': 'Suite 198', 'city': 'Lebsackbury', 'zipcode': '31428-2261', 'geo': {'lat': '-38.2386', 'lng': '57.2232'}} | 024-648-3804 | ambrose.net | {'name': 'Hoeger LLC', 'catchPhrase': 'Centralized empowering task-force', 'bs': 'target end-to-end models'} |

Agora sabemos como utilizar a biblioteca requests e o módulo JSON para obter dados de APIs!