05

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:

```
is 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
```

COPIAR CÓDIGO

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**:

```
_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)
```

COPIAR CÓDIGO

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',
          }
      }
```

COPIAR CÓDIGO

Fazendo a normalização:

```
pd.json_normalize(json_obj)
```

COPIAR CÓDIGO

Sexo_biologico	Saude.Dificuldade_caminhar	Saude.Atividade_fisica	Saude.IMC	Saude.Doenca_cardiaca
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'}
       }
       ]
```

COPIAR CÓDIGO

Fazendo a normalização:

```
pd.json_normalize(json_list)
```

COPIAR CÓDIGO

index	ID	Faixa_etaria	Sexo_biologico	Saude.Dificuldade_caminhar	Saude.Atividade_fisica	Saude.I
0	01	55-59	Feminino	Nao	Sim	16.6
1	02	80 ou +	Feminino	Nao	Sim	20.34

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)](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"
```

```
        ]
      }
    ]
}
```

COPIAR CÓDIGO

Se tentarmos normalizar esses dados:

```
pd.json_normalize(dados_dict)
```

COPIAR CÓDIGO

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'])
```

COPIAR CÓDIGO

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'])
```

COPIAR CÓDIGO

Com ambos os códigos o resultado é o mesmo.

Sexo_biologico	Raça	...	Atividade_fisica	Saude_geral	Horas_sono	Problemas_saude
Feminino	Branca	...	Sim	Muito boa	5	Diabetes,Asma,Cancer_pele
Feminino	Branca	...	Sim	Muito boa	7	AVC
Masculino	Branca	...	Sim	Muito boa	8	diabetes,Asma

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
0	01	55-59	Feminino	...	[Diabetes,Asma,Cancer_pele]	Principais Indicadores de D
1	02	80 ou +	Feminino	...	AVC	[Principais Indicadores de I
2	03	65-69	Masculino	...	[diabetes,Asma]	Principais Indicadores de D

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', 'A
```

COPIAR CÓDIGO

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