

Trabalhar com dados é uma arte e uma ciência. Temos falado principalmente sobre a parte científica, mas neste capítulo veremos um pouco da arte.

# Manipulando Dados

Uma das habilidades mais importantes de um cientista de dados é a manipulação de dados. É mais uma abordagem geral do que uma técnica específica, então vamos trabalhar com alguns exemplos. Sempre haverá um desafio diferente. Assim, o ideal é que você aprenda as técnicas de forma geral para saber aplicar em cada circunstância. Vamos resgatar alguns exemplos vistos nas semanas anteriores. Lá, vimos que alguns dados vêm 'sujos', 'bagunçados' ou com informações que são desnecessárias. Lembre-se que no mundo do *Big Data*, temos uma massa de dados gigantesca e que chega em uma velocidade muito rápida... Então, cada "espacinho" de memória que salvamos será importante na eficiência computacional de nossos algoritmos.

## Removendo dados faltantes

No exemplo dos filmes da *Pixar*, ainda não tínhamos informações quanto ao valor de bilheteria do filme *Soul*. Portanto, gostaríamos de, por hora, remover essa linha de nossas tabelas:

```
In [ ]: import pandas as pd  
  
dados_pixar = pd.read_csv("/content/dados_pixar.csv", sep = ";")  
dados_pixar.dropna()
```

Podemos repetir o mesmo para o conjunto de informações das personagens de *Star Wars*. Note que neste conjunto, é ainda mais complicado pois há valores faltantes em diferentes colunas. Através deste método, ele elimina automaticamente todas as linhas que contém *ao menos* um valor faltante:

```
In [ ]: dados_starwars = pd.read_excel("/content/dados_starwars.xlsx")  
  
dados_starwars.dropna()
```

Note que o `Data Frame` original permanece o mesmo. Ele somente se altera quando o salvamos da seguinte forma:

```
In [3]: # antes de salvar com alteracao  
print(dados_starwars.shape)  
print(dados_starwars.dropna().shape)  
  
# apos salvar as alteracoes  
dados_starwars = dados_starwars.dropna()  
print(dados_starwars.shape)  
print(dados_starwars.dropna().shape)
```

```
(87, 8)
(50, 8)
(50, 8)
(50, 8)
```

Ou seja, saímos de uma tabela com 87 linhas e 8 colunas para uma com 50 linhas e as mesmas 8 colunas. As 37 linhas que continham valores faltantes foram excluídas.

No mundo real, informações faltantes são comuns. Nem sempre podemos removê-las. Existe toda uma ciência estatística de como tratá-las mas que está fora do escopo deste curso.

## Selecionando linhas e colunas específicas

A manipulação de dados é essencial durante a raspagem de dados. Muitas vezes (se não todas), a tradução do `HTML` para uma tabela no Python não é perfeitamente realizada e assim, abre-se o espaço para usarmos nosso talento com as técnicas de organização e limpeza da base de dados. Retorne ao exemplo da raspagem de dados sobre o Rio de Janeiro.

```
In [23]: import requests
from bs4 import BeautifulSoup
from io import StringIO

# retirando o texto do HTML de uma pag na web
url = requests.get('https://pt.wikipedia.org/wiki/Rio_de_Janeiro_(estado)').text
busca = BeautifulSoup(url, 'lxml')
tabela = busca.find_all('table')
```

```
In [24]: # tab 1: crescimento populacional
cresc_pop = pd.read_html(StringIO(str(tabela)))[6]
```

Temos 2 problemas: Na linha 13, o conteúdo não condiz com o restante da tabela. E a coluna 3 é um 'lixo' que está atrapalhando nossa análise. Vamos removê-las:

```
In [25]: # selecionando Linhas 1 ate 13 (indices 0 ate 13)
cresc_pop_nova = cresc_pop.iloc[0:13, :]

# selecionando colunas 1, 2 e 4 (indices 0, 1 e 3)
cresc_pop_nova = cresc_pop_nova.iloc[:, [0, 1, 3]]
```

## Renomeando colunas (ou variáveis)

Durante as raspagems, os nomes acabaram se misturando e não ficando algo muito comprehensível. Podemos alterá-los usando o seguinte comando:

```
In [26]: # nomes anteriores
print(cresc_pop_nova.columns)

# novos nomes
cresc_pop_nova.columns = ['ano', 'populacao', 'crescimento_percentual']
print(cresc_pop_nova.columns)
```

```
MultiIndex([('Crescimento populacional', 'Censo'),
            ('Crescimento populacional', 'Pop.'),
            ('Crescimento populacional', '%±')],
           )
Index(['ano', 'populacao', 'crescimento_percentual'], dtype='object')
```

Existem várias dicas de 'boas práticas' sobre nomenclaturas de objetos na linguagem de programação Python que não iremos entrar muito em detalhes. Em geral, evitaremos letras maiúsculas e minúsculas conjuntamente, espaços e acentos.

## Verificando e alterando os tipos das variáveis

Analisar e verificar os tipos das variáveis é muito importante. Suponha que uma variável que naturalmente é numérica esteja classificada como texto e você necessite realizar um cálculo com ela. O Python ficará confuso e pode ser até que ele realize a tarefa mas entregará um resultado totalmente inesperado. Vamos verificar e modificar isso nesta tabela:

```
In [27]: # antigo tipos
print(cresc_pop_nova.dtypes)
```

ano	object
populacao	object
crescimento_percentual	object
dtype:	object

```
In [28]: # novos tipos

## transforma para tipo 'int'
cresc_pop_nova["ano"] = cresc_pop_nova["ano"].astype("int")

## transforma para tipo 'int'
# remove os espaços em branco
# transforma para tipo 'int'
cresc_pop_nova["populacao"] = (cresc_pop_nova["populacao"]
                                 .str.replace(u'\xa0', u' ')
                                 .astype("int"))

## transforma para tipo 'float'
# remove %
# substitui ',' por '.'
cresc_pop_nova["crescimento_percentual"] = (cresc_pop_nova["crescimento_percentual"]
                                              .str.replace('%', '')
                                              .str.replace(',', '.')
                                              .astype("float"))
```

Esclarecendo que o tipo `object` é algo que mistura texto com números. Isso pode confundir na hora de fazer um cálculo, por exemplo. Além disso, existem diversos tipos diferentes no Python. Vamos nos concentrar somente nos principais:

- `integer`: Formato numérico sem casas decimais. Ex.: `3.14159265359` é entendido pelo computador como `3`;
- `float`: Formato numérico com casas decimais. Ex.: `1` é entendido pelo computador como `1.00000000...`;

- `string` : Formato de texto. Ex.: Mesmo um número como `2023` é compreendido pelo computador como um texto similar a `texto` .

Agora podemos realizar diversos cálculos com as variáveis numéricas.

## Criando colunas (ou variáveis)

Vamos criar uma coluna para inserir a informação do quanto a população cresceu de um registro para outro (em valor bruto!) usando os seguintes comandos:

```
In [30]: cresc_pop_nova["crescimento_pop"] = cresc_pop_nova['populacao'].diff()
```

## Filtrando colunas (ou variáveis)

Por fim, muitas vezes temos interesse de parte dos nossos dados. Por exemplo, em uma base gigante de clientes, podemos ter interesse somente naqueles com idade maior ou igual a 40 anos ou somente pessoas do sexo feminino... No nosso caso, vamos filtrar somente os dados a partir de 1950 da seguinte maneira:

## Reordenando colunas (ou variáveis)

```
In [32]: cresc_pop_nova = cresc_pop_nova[cresc_pop_nova["ano"] >= 1950]
```

Gostaríamos que a variável `crescimento_pop` fosse a terceira em nossa tabela. Podemos simplesmente ordenar nossa tabela usando algo que já aprendemos por aqui:

```
In [34]: cresc_pop_nova = cresc_pop_nova.iloc[:, [0, 1, 3, 2]]
```

Por fim, esta é a nossa tabela limpa e organizada!

```
In [36]: cresc_pop_nova
```

```
Out[36]:
```

	ano	populacao	crescimento_pop	crescimento_percentual
5	1950	4674645	1062647.0	29.4
6	1960	6709891	2035246.0	43.5
7	1970	9110324	2400433.0	35.8
8	1980	11489797	2379473.0	26.1
9	1991	12783761	1293964.0	11.3
10	2000	14367083	1583322.0	12.4
11	2010	15989929	1622846.0	11.3
12	2022	16055174	65245.0	0.4

Ao longo do curso (e da vida), faremos esses tipos de manipulação de forma natural. Com o tempo, você será um(a) craque nesses procedimentos.

# Explorando Dados

Depois de identificar as perguntas que você está tentando responder e colocar as mãos em alguns dados, você pode ficar tentado mergulhar e começar imediatamente a construir modelos e obter respostas. No entanto, seu primeiro passo deve ser explorar seus dados.

## Explorando Dados Unidimensionais

O caso mais simples é quando você tem um conjunto de dados unidimensional, que é apenas uma coleção de números. Por exemplo, pode ser o número médio diário de minutos que cada usuário gasta em seu site, o número de vezes que cada vídeo de uma coleção de tutoriais foi assistido ou o número de páginas de cada livro de uma biblioteca. Um primeiro passo óbvio é calcular algumas estatísticas resumidas. Você gostaria de saber quantos pontos de dados você tem, o menor, o maior, a média e o desvio padrão. Ainda assim, não lhe dá uma grande compreensão sobre os dados. Um bom próximo passo é criar visualizações como um histograma ou um gráfico de barras. Vamos recuperar os dados sobre Estados utilizando a API do [IBGE](#) como vimos anteriormente. Lembrando que precisamos instalar essa biblioteca virtualmente:

```
In [37]: !pip install ibge
Collecting ibge
  Downloading ibge-0.0.5-py3-none-any.whl.metadata (18 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from ibge) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->ibge) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->ibge) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->ibge) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->ibge) (2024.8.30)
  Downloading ibge-0.0.5-py3-none-any.whl (6.4 kB)
Installing collected packages: ibge
Successfully installed ibge-0.0.5
```

Feito isso, encontramos os dados sobre os Estados brasileiros da seguinte forma:

```
In [38]: import pandas as pd
import matplotlib.pyplot as plt
from ibge.localidades import *

estados = Estados()
df_estados = pd.json_normalize(estados.json())
```

Vamos obter a quantidade de Estados em cada região do país:

```
In [39]: # quantidade de regioes
print(df_estados["regiao.sigla"].nunique())

# nome de cada regiao
print(df_estados["regiao.nome"].unique())
```

```
# numero de vezes que cada região aparece
# ou seja, o numero de Estados em cada região
print(df_estados["regiao.nome"].value_counts())

5
['Norte' 'Nordeste' 'Sudeste' 'Sul' 'Centro-Oeste']
regiao.nome
Nordeste      9
Norte         7
Sudeste       4
Centro-Oeste  4
Sul           3
Name: count, dtype: int64
```

Vamos recuperar os dados sobre o valor *real* do salário mínimo como também vimos anteriormente. Lembrando que também precisamos instalar essa biblioteca virtualmente:

In [40]: `!pip install ipeadatapy`

```
Collecting ipeadatapy
  Downloading ipeadatapy-0.1.9-py3-none-any.whl.metadata (297 bytes)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
  (from ipeadatapy) (2.2.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
  (from ipeadatapy) (2.32.3)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-pac
kages (from pandas->ipeadatapy) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.1
0/dist-packages (from pandas->ipeadatapy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pack
ages (from pandas->ipeadatapy) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-pa
ckages (from pandas->ipeadatapy) (2024.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.
10/dist-packages (from requests->ipeadatapy) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pack
ages (from requests->ipeadatapy) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dis
t-packages (from requests->ipeadatapy) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dis
t-packages (from requests->ipeadatapy) (2024.8.30)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages
  (from python-dateutil>=2.8.2->pandas->ipeadatapy) (1.16.0)
  Downloading ipeadatapy-0.1.9-py3-none-any.whl (10 kB)
Installing collected packages: ipeadatapy
Successfully installed ipeadatapy-0.1.9
```

## OBSERVAÇÃO:

*Valor real* do salário mínimo significa que ele foi *defacionado*.

Ou seja, foi feito um cálculo na tentativa de remoção da inflação ao longo dos anos de forma que os valores do passado sejam comparáveis de forma direta com os mais recentes.

In [42]: `import ipeadatapy`

```
# o código da série de salário mínimo na base
# e 'GAC12_SALMINRE12'
df_salmin = ipeadatapy.timeseries('GAC12_SALMINRE12')

# Visualizando os dados (somente o final)
df_salmin.tail()
```

Out[42] :

	CODE	RAW DATE	DAY	MONTH	YEAR	VALUE (R\$ (do último mês))
DATE						
2024-05-01	GAC12_SALMINRE12	2024-05-01T00:00:00-03:00	1	5	2024	1424.028341
2024-06-01	GAC12_SALMINRE12	2024-06-01T00:00:00-03:00	1	6	2024	1420.476746
2024-07-01	GAC12_SALMINRE12	2024-07-01T00:00:00-03:00	1	7	2024	1416.792411
2024-08-01	GAC12_SALMINRE12	2024-08-01T00:00:00-03:00	1	8	2024	1418.778027
2024-09-01	GAC12_SALMINRE12	2024-09-01T00:00:00-03:00	1	9	2024	1412.000000

## Duas Dimensões

Agora imagine que você tem um conjunto de dados com duas dimensões. Claro que você gostaria de entender cada dimensão individualmente. Mas você pode analisar ambos de forma conjunta. Isso será bem importante nas próximas semanas.

Por exemplo, considere um clássico conjunto de dados descrevendo o peso (em libras) e altura (em polegadas) de adultos saudáveis:

In [44] :

```
import requests
from bs4 import BeautifulSoup
from io import StringIO

# retirando o texto do HTML de uma pag na web
url = requests.get('http://socr.ucla.edu/docs/resources/SOCR_Data/SOCR_Data_Dinov_6')
busca = BeautifulSoup(url, 'lxml')
tabela = busca.find_all('table')

# tabela com as medidas antropometricas
# Height(inches) = Altura(polegadas)
# Weight(pounds) = Peso(libras)
medidas_antro = pd.read_html(StringIO(str(tabela)), header = 0)[0]
```

In [45] :

```
# removendo a primeira coluna
medidas_antro = medidas_antro.iloc[:, 1:3]

# alterando os nomes para portugues
medidas_antro.columns = ['altura', 'peso']

# transformando altura de polegadas para cm
medidas_antro["altura"] = medidas_antro["altura"] * 2.54

# transformando peso de libras para kg
medidas_antro["peso"] = medidas_antro["peso"] / 2.205
```

Em geral, conforme a altura da pessoa aumenta, seu peso também aumenta (o que faz todo sentido físico). Embora, a relação não seja perfeita para todos e envolva outros aspectos que analisaremos mais a frente.

# Muitas Dimensões

Com muitas dimensões, você tem interesse em saber como todas se relacionam entre si. Porém, não é nada trivial realizar essa tarefa, uma vez que, como humanos, já temos muita dificuldade em entender figuras tridimensionais. Além disso, nesta realidade, não conseguimos ilustrar imagem com 4 ou mais dimensões. E no mundo real, seu problema terá muitas dimensões. Mas não se preocupe, vamos por partes.

Vamos utilizar como exemplo um banco de dados clássico que é bem útil para o treinamento de modelos de aprendizado de máquina. Veremos isso nas próximas semanas. Especificamente, este banco de dados trata de informações coletadas sobre diferentes flores de Íris. Não sabe nada sobre flores? A figura abaixo ilustra um breve resumo do que precisamos saber para entender do que os dados tratam.



Por ser muito utilizado, a biblioteca `scikit-learn` disponibiliza esse banco de forma direta através dos seguintes comandos:

```
In [ ]: from sklearn import datasets

iris = datasets.load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df["especie"] = iris.target

# renomeando variaveis
# compr_sep: comprimento da sepala
# larg_sep: Largura da sepala
# compr_pet: comprimento da petala
# larg_pet: Largura da petala
iris_df.columns = ['compr_sep', 'larg_sep', 'compr_pet', 'larg_pet', 'especie']

# reordenando
iris_df = iris_df.iloc[:, [4, 0, 1, 2, 3]]

# reclassificando
iris_df.loc[iris_df["especie"] == 0, "especie"] = "Setosa"
iris_df.loc[iris_df["especie"] == 1, "especie"] = "Versicolor"
iris_df.loc[iris_df["especie"] == 2, "especie"] = "Virginica"
```

Temos 5 variáveis, sendo 4 delas medidas de comprimento e largura da sépala e pétala de 150 flores de Íris de 3 espécies distintas: **0** representa *Setosa*, **1** representa *Versicolor* e **2** representa *Virginica*.

Não gosta de flores? Calma! Vamos ver um exemplo prático com dados reais do mercado financeiro. O [Yahoo! Finanças](#) disponibiliza diversos dados sobre mercado financeiro como a evolução de investimentos, ações na Bolsas de valores e criptomoedas. De forma arbitrária, selecionamos 5 grandes ações brasileiras negociáveis na Bolsa de valores:

- **B3SA3.SA**: B3 S.A. - Brasil, Bolsa, Balcão
- **BBAS3.SA**: Banco do Brasil S.A.
- **ELET3.SA**: Centrais Elétricas Brasileiras S.A. - Eletrobras
- **EMBR3.SA**: Embraer S.A.
- **PETR4.SA**: Petróleo Brasileiro S.A. - Petrobras

O Python também desfruta de uma biblioteca inteiramente dedicada à extrair dados deste site e nos disponibilizar de forma totalmente organizada. Obtemos todos os dados relativos ao ano de 2022 através dos seguintes comandos:

```
In [ ]: !pip install yfinance  
import yfinance as yf
```

```
In [ ]: # B3SA3.SA: B3 S.A. - Brasil, Bolsa, Balcão  
# BBAS3.SA: Banco do Brasil S.A.  
# ELET3.SA: Centrais Elétricas Brasileiras S.A. - Eletrobrás  
# EMBR3.SA: Embraer S.A.  
# PETR4.SA: Petróleo Brasileiro S.A. - Petrobras  
acoes_br = yf.download("B3SA3.SA BBAS3.SA ELET3.SA EMBR3.SA PETR4.SA",  
                        start = "2022-01-01",  
                        end = "2022-12-31")
```

```
In [ ]: # separando somente os valores de fechamento  
acoes_br = acoes_br.iloc[:, 5:10]  
  
# renomeando  
acoes_br.columns = ['B3SA3.SA', 'BBAS3.SA', 'ELET3.SA', 'EMBR3.SA', 'PETR4.SA']  
acoes_br.index.name = 'Date'
```

## Referências Adicionais

- [Pandas](#) é uma das principais ferramentas Python para limpar, organizar, manipular e trabalhar com dados.
- Se você tem interesse pelo mercado financeiro, explore o [Yahoo! Finanças](#).