SOFTEX/UNIFEI - Curso de Introdução à Inteligência Artificial

Módulo 3. Aprendizado de Máquina

Aula 05 - Tutorial Notebook 1 - Introdução ao Aprendizado de Máquina

Assunto: Fases Iniciais do Processo de Machine Learning

Autor: Rafael Frinhani

Tópico: Processo de *Machine Learning*.

Objetivo: Este notebook visa a apresentação de procedimentos que podem ser usados para a coleta, entendimento (exploração e descrição) e limpeza de conjuntos de dados. Tais atividades são comumente realizadas nas fases de iniciais do processo de *Machine Learning*, auxiliando na verificação da qualidade dos dados e sua preparação para execução dos modelos de aprendizado.

Organização do Notebook: Os exemplos a seguir correspondem as fases iniciais do processo de *Machine Learning:* Coleta de Dados, Entendimento dos Dados e Limpeza dos Dados.

Preparação do Ambiente: Para a correta apresentação dos exemplos deste notebook é necessário instalar a biblioteca *pandas-profiling*, útil para Análise Exploratória de Dados e será usada para gerar as descrições dos dados. A instalação ou atualização de bibliotecas é por vezes adotada para que seja considerada no código uma versão mais atual que a biblioteca padrão do Google Colab.

A instalação da *pandas-profiling* é feita através do sistema de gerenciamento de pacotes *pip* pelo comando a seguir:

```
!pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
```

Os comandos à seguir correspondem a importação das principais bibliotecas necessárias para os exemplos deste notebook:

```
import numpy as np # Manipulação de matrizes
import pandas as pd #
from lxml import objectify # Permite usar XML como se fosse um objeto Python
import requests as req # Comunicação via Http
from bs4 import BeautifulSoup # Permite extrair dados de arquivos HTML e XML (parse).
from pandas_profiling import ProfileReport # Análise Exploratória de dados
from google.colab import drive # Ler e salvar arquivos do Google Drive
drive.mount("/content/drive/") # Montagem da pasta do drive
```

Conforme o Processo de Machine Learning, as fases iniciais correspondem a Coleta de Dados, Entendimento dos Dados e Limpeza dos Dados.

---

1. Coleta de Dados

A Coleta de Dados pode ser feita manualmente ou automaticamente (ex. sensores), sendo que o acesso aos dados é o primeiro passo da fase de entendimento. As operações de entrada e saída de dados são geralmente realizadas através da leitura de arquivos-texto e outros formatos diretamente em disco (ex. txt, xml, csv, json etc), bancos de dados (ex. PostgreSQL, Oracle, MongoDB etc) e fontes de dados em rede (ex. APIs, html, páginas web etc). A seguir são apresentados exemplos de coleta de dados baseados em arquivo e web.

1. Arquivo Local:

Caso possua o arquivo de um conjunto de dados no equipamento local, é possível realizar seu *upload* em uma pasta do Google Colab pelo menu lateral esquerdo "Arquivos". Para descobrir o caminho completo do arquivo clique com o botão direito do mouse sobre ele e selecione "Copiar Caminho".

O arquivo do conjunto de dados deve ser armazenado em uma estrutura de dados que permita a sua manipulação a nível de código. A biblioteca Pandas (https://pandas.pydata.org/) é comumente utilizada para análise de dados e possibilita o trabalho em dados estruturados ou tabulares.

Com Pandas, o conteúdo do arquivo será armazenado em um *DataFrame*, que é uma estrutura de dados tabular, orientada a colunas, com rótulos para as linhas e para colunas. O código a seguir importa as bibliotecas e faz a carga de um arquivo de um conjunto de dados em um *DataFrame*.

```
# Bibliotecas necessárias: pandas, pandas_profiling
# Carga do arquivo .csv localizado na pasta 'content' para o DataFrame (df). O parâmetro 'sep' define o caracter separador das colunas do
df = pd.read_csv("/content/dataset-credit_card.csv", sep=';') # use header=None para desconsiderar os cabeçalhos
print(df)
```

---

2. Arquivo na Web:

Também é possível fazer a carga de um arquivo localizado na web. No exemplo a seguir são coletados os dados de um catálogo de plantas disponível em um arquivo XML em http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/xml/plant_catalog.xml. **XML (*eXtensible Markup Language*)** é um formato estruturado de dados, que aceita a criação de hierarquias e aninhamento com metadados.

A biblioteca `lxml` pode ser usada para manipular esse tipo de arquivo, sendo feito um parse do documento XML de modo a se obter uma referência para a tag raiz do arquivo:

```
#from lxml import objectify (Biblioteca necessária)

dados_url = 'http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/xml/plant_catalog.xml'
dados_parsed = objectify.parse(dados_url)
tag_raiz = dados_parsed.getroot()
```

> Uma vez descoberto a tag raiz do arquivo é possível acessar o gerador das tags PLANT, que produz cada elemento do XML que armazena dados de uma planta. Para cada registro do arquivo XML é obtido um dicionário com os rótulos das tags dos atributos e os valores dos dados:

```
dados_plantas = []

for elt in tag_raiz.PLANT: # Lê cada tag filho da tag raiz
  el_dado = {}
  for filho in elt.getchildren():
    el_dado[filho.tag] = filho.pyval
  dados_plantas.append(el_dado)
dados_plantas
```

> Os dados constantes no dicionário são convertidos em um *DataFrame*, conforme comando a seguir:

```
plantas = pd.DataFrame(dados_plantas) # converte a lista com os dados para o DataFrame plantas
plantas.head() # por padrão mostra os cinco primeiros registros
```

---

3. API Web:

Muitos sites disponibilizam APIs públicas (*Application Programming Interface*, Interface de Programação de Aplicação) que oferecem fluxos de dados via arquivo (ex. formato JSON). Uma das maneiras de acessar uma API utiliza o comando `request`.

No exemplo a seguir é consultada uma API do GitHub que retorna trinta das questões (issue) mais recentes do repositório da biblioteca `pandas`.

```
import requests

url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
url = requests.get(url)
dados_url = url.json()
questoes = pd.DataFrame(dados_url, columns=['number', 'title', 'labels', 'state'])
questoes
```

---

4. Página Web (*Crawler*):

Um dos modos de coleta de dados envolve obter o conteúdo diretamente de páginas html. Esse procedimento pode ser feito em procedimentos como *Web Scrapping* e *Web Crawling*. Tabelas são as estruturas mais comuns para disponibilização de dados estruturados.

Como exemplo, o site de uma agência independente dos Estados Unidos chamada Federal Deposit Insurance Corporation (FDIC), contém dados de falência de bancos (https://www.fdic.gov/resources/resolutions/bank-failures/failed-bank-list/), que podem ser coletados da seguinte mandeira:

```
# Função para coletar dados tabulados de páginas HTML e armazenar numa estrutura DataFrame
dados_site = pd.read_html('https://www.fdic.gov/resources/resolutions/bank-failures/failed-bank-list/')
tabela_falencias = dados_site[0] #o site é composto de uma única tabela, a qual é extraída
tabela_falencias.head()
```

Situações que requerem *parsing*: Em alguns sites pode ser que os dados tabelados coexistam com outros dados como títulos, conteúdo descritivo, imagens etc. Os dados coletados diretamente de um website possuem um propósito visualização, não estando no formato adequado para manipulações.

Nestes casos, pode ser necessário realizar o *parsing* dos dados, que é o processo de analisar um texto para determinar sua estrutura lógica. A biblioteca *BeautifulSoup* possibilita que se faça o *parsing* para extração de dados de arquivos HTML e XML.

O exemplo a seguir utiliza o conteúdo disponível no site Worldmeters (www.worldometers.info/coronavirus), que contém dados de COVID19. Os dados foram coletados corretamente se a resposta para o comando print(dados_html) é igual a "<Response [200]>".

```
#import requests as req # Acesso via Http
#from bs4 import BeautifulSoup # Permite extrair dados de arquivos HTML e XML (parse).
#import numpy as np
#import pandas as pd

# Dados são coletados da página web
dados_html = req.get("https://www.worldometers.info/coronavirus")
```

> Uma vez que a página foi coletada com sucesso, seu conteúdo pode ser visualizados pelo comando `content`. Caso executado, nota-se que os dados não estão em formato adequado, sendo a necessária a realização do *parsing* com a biblioteca *BeautifulSoup* conforme a seguir:

```
dados_html_parsed = BeautifulSoup(dados_html.content)
dados_html_parsed
```

> A partir dos dados parseados é possível localizar os que são de interesse e armazená-los em uma tabela que possibilite sua manipulação. Para isso, a função `html_parsed.find` busca por termos parseados conforme um dado atributo.

> Embora os dados ainda não estejam em um formato adequado, é possível identificar que a linha de índice 0 contém os rótulos dos atributos observados na tabela disponível no site.

All | Europe | North America | Asia | South America | Africa | Oceania

#	Country, Other	Total Cases	New Cases	Total Deaths	New Deaths	Total Recovered	New Recovered	Active Cases	Serious, Critical
	World	675,583,121	+115,454	6,765,923	+799	647,924,317	+145,826	20,892,881	41,897
1	USA	104,289,430		1,134,259		101,378,824		1,776,347	3,425
2	India	44,682,895		530,740		44,150,372		1,783	698
3	France	39,528,817		164,286		*39,271,652*		92,879	869
4	Germany	37,796,790		165,865		37,413,200	+15,100	217,725	1,281
5	Brazil	36,837,943		697,200		35,933,396		207,347	8,318
6	Japan	32,633,741	+45,299	68,796	+397	21,576,391	+8,966	10,988,554	475
7	S. Korea	30,213,928	+16,862	33,522	+36	*29,773,405*	+32,528	407,001	345

```
tabela = dados_html_parsed.find('table', attrs={'id': 'main_table_countries_today'})
tabela
linhas = tabela.find_all('tr')
linhas[0]
```

> O acesso ao primeiro registro dos dados pode ser feito pelo comando a seguir:

```
linhas[0].text.strip()
linhas[9].text.strip().split("\n")
```

> Os dados podem ser salvos de forma permanente em um arquivo. No exemplo a seguir, em um primeiro momento as linhas e colunas de interesse dos dados parseados são salvas em uma lista. A lista é então convertida em um *DataFrame* Pandas com os dados relevantes.

```
#from google.colab import drive
#drive.mount("/content/drive/")

data_File = [] # lista que armazena os dados que serão salvos em arquivo

# Lê cada linha dos dados parseados e as adiciona em uma lista, incluindo as colunas de id 1 a 10.
for linha in linhas:
    data_File.append(linha.text.strip().split("\n")[1:10])

df = pd.DataFrame(data_File[9:], columns=data_File[0]) # Os dados relevantes iniciam na linha 9 do arquivo.
df.to_csv("/content/drive/MyDrive/Colab Notebooks/dataset-covid19.csv") # Armazena o DataFrame em um arquivo .csv no Google Drive do usuá
df
```

---

2. Entendimento dos Dados

Esta fase tem como principal atividade a descrição dos dados. Os exemplos a seguir consideram um conjunto de dados, cujo cenário abordado é o de uma empresa de serviços financeiros que tem interesse em amenizar os impactos da inadimplência do pagamento de cartão de crédito. A empresa deseja prever uma inadimplênciaa considerando os dados pessoais e histórico financeiro do cliente. Para um melhor entendimento do negócio consulte as informações disponíveis no UCI Machine Learning Repository. Procure entender o problema que a empresa está enfrentando e como ele foi tratado pelos autores.

O conjunto de dados que será utilizado é uma adaptação para propósitos didáticos do conjunto original disponibilizado pela empresa no repositório da UCI. Uma vez com o arquivo do conjunto de dados em mãos (*dataset-credit_card.csv*) faça o seu upload na pasta *'\content'* do Google Colab.

Após armazenar o conjunto de dados em um *DataFrame* é possível obter informações sobre ele *DataFrame df* pode ser obtida pelo comando `shape`, que retorna uma tupla com a quantidade de linhas e de colunas:

```
#import pandas as pd # importando pandas
#from pandas_profiling import ProfileReport # importando pandas-profiling

# Bibliotecas necessárias: pandas, pandas_profiling
# Carga do arquivo .csv localizado na pasta 'content' para o DataFrame (df). O parâmetro 'sep' define o caracter separador das colunas do
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/dataset-credit_card.csv", sep=';') # use header=None para desconsiderar os cabeç
print(df)
```

```
# Dimensão do DataFrame df
df.shape
```

O comando `columns` que retorna os rótulos das colunas do *DataFrame*. Acesse o link do conjunto de dados para mais informações sobre os atributos representados por cada coluna.

```
# Rótulos das colunas do DataFrame df
df.columns
```

É possível identificar o tipo de dados de cada atributo e se existem valores ausentes através do comando `info`. Conforme a seguir, o resultado mostra que 23 colunas armazenam dados do tipo inteiro (int64) e 2 colunas armazenam dados do tipo *object*, que representa um tipo de "objeto arbitrário" ou cadeias de caracteres (*strings*). O valor 30000 na coluna "Non-Null Count" indica que todas as colunas de todos os registros possuem algum valor, mas atenta-se para o fato que podem existir valores inválidos.

```
# Informações do tipo de cada atributo e existência de valores ausentes
df.info()
```

Para se ter uma noção dos valores em cada coluna, a função `head` permite visualizar alguns registros e atributos do conjunto. Por padrão a biblioteca mostra os cinco primeiros registros, mas essa quantidade pode ser alterada via parâmetro. A depender da quantidade algumas colunas poderão ser ocultadas para uma melhor visualização.

```
df.head()
```

Uma descrição mais detalhada pode ser obtida através da função `describe`, que retorna uma estatística descritiva do conjunto de dados, resumindo a tendência central e dispersão. Fornece contagens, valores únicos e valores mais frequentes, média, desvio padrão, limites mínimos e máximo de valores por atributo e as distribuições por quartis.

```
# O parâmetro include='all' é necessário para descrever os atributos categóricos (object)
# Sem include, a função descreve apenas os dados numéricos por padrão.
df.describe(include='all')
```

Uma visualização mais amigável para descrição dos dados é possível com a biblioteca *pandas-profiling*, que gera relatórios HTML iterativos a partir de um *DataFrame*, possibilitando a análise exploratória de dados. A construção do relatório pode demorar dependendo do tamanho do conjunto de dados.

```
#import pandas as pd # importando pandas
#from pandas_profiling import ProfileReport # importando pandas-profiling

#Define a a fonte de dados do relatório, título e estilo do HTML gerado
```

```
profile = ProfileReport(df, title='RELATÓRIO DO CONJUNTO DE DADOS', html={'style':{'full_width':True}})
profile.to_notebook_iframe()
```

O relatório gerado pode ser exportado como um arquivo .html, podendo ser utilizado fora do ambiente Web através de um visualizador .html ou browser.

```
profile.to_file(output_file="descrevendo_dados.html")
```

3. Limpeza de Dados

No conjunto de dados usado como exemplo, alguns atributos não têm valores correspondentes com os que foram informados na descrição do conjunto de dados. É o caso do atributo PAY_1 que tem dois valores não documentados (0 e -2), além de uma *string* 'Not available' indicando dados ausentes. Para remover do conjunto de dados os registros com valores 'Not available', incialmente é feita uma busca de todos os valores diferentes deste rótulo:

```
mascara_pay1_validos = df['PAY_1'] != 'Not available' # Encontra as linhas cujo valor é diferente de 'Not available'
sum(mascara_pay1_validos)
```

```
mascara_pay1_validos
```

O resultado mostra que 26.979 registros não possuem o valor 'Not available', indicando que 3.021 possuem ausência de dados (30.000 - 26.979 = 3.021). Para realizar a limpeza desses registros e salvar o resultado em uma cópia do *DataFrame*:

```
df_limpo_1 = df.loc[mascara_pay1_validos,:].copy() # Usa a máscara e seleciona os registros válidos
df_limpo_1['PAY_1'].value_counts() # Contagem dos valores após a limpeza
```

```
df.dtypes
```

Embora não esteja previsto no conjunto de dados original, considere que os valores -2 e 0 foram inseridos para representar outras situações do pagamento. O valor -2 representa uma conta que começou o mês sem valor a ser pago e o crédito não foi usado. O valor 0 significa que o pagamento mínimo foi feito, mas o saldo total devedor não foi pago.

Como o atributo não possui mais os rótulos 'Not available', é necessário transformar o tipo do atributo para numérico, uma vez que ela foi importada pelo Pandas como do tipo 'object' devido os valores textuais. O método `astype()` permite a conversão do tipo de dados:

```
df_limpo_1['PAY_1'] = df_limpo_1['PAY_1'].astype('int64') # Converte o tipo da
df_limpo_1[['PAY_1']].info() # Exibe as informações de uma coluna
```

Outra informação que foi obtida a partir da exploração dos dados é a existência de valores repetidos do atributo ID. Para solucionar este problema, inicialmente é verificada a quantidade de valores únicos na coluna ID com a função `nunique`:

```
df_limpo_1['ID'].nunique() # Verifica valores únicos na coluna
```

O valor acima indica que dos 26.797 registros resultantes da limpeza feita anteriormente, 26.704 possuem valor de ID único. Com isso, constata-se que existe uma duplicação de IDs, mas não se sabe ao certo se existem diversos IDs com valores duplicados ou se um único ID está sendo duplicado várias vezes. O método `value_counts()` é útil para obter melhores informações sobre as duplicidades:

```
contagem_id = df_limpo_1['ID'].value_counts() # Conta as ocorrências dos valores de uma coluna
contagem_id.head() # Por padrão exibe os 5 primeiros registros
```

A variável 'contagem_id' armazena o número de vezes que um ID apareceu no conjunto. Para obter a contagem dessas repetições, a operação é repetida considerando agora essa variável:

```
contagem_id.value_counts()
```

```
1    26429
2      275
Name: ID, dtype: int64
```

O resultado da contagem mostra que a maioria dos IDs ocorre uma única vez, sendo que 275 IDs ocorrem duas vezes. Nenhum ID ocorre mais de duas vezes. Sendo assim, cria-se uma máscara booleana para acessar os índices dos registros que possuem contagem igual a 2:

```
mascara_duplicados = contagem_id == 2 # Cria uma máscara lógica onde o valor de ocorrências for igual a 2
mascara_duplicados[0:5] # Mostra os 5 primeiros registros de IDs duplicados
```

Com a máscara criada, os IDs que estão duplicados são selecionados:

```
ids_duplicados = contagem_id.index[mascara_duplicados] # Retorna os IDs dos registros que estão na máscara
ids_duplicados = list(ids_duplicados) # Converte para uma lista
len(ids_duplicados) # Verifica o tamanho da lista
```

A lista de IDs duplicados contém 275 itens, que corresponde com a quantidade obtida nas análises anteriores. Essa lista será usada para examinar os registros que possuem IDs duplicados.

Solucionando Duplicidades

Utilizando os primeiros 3 IDs da lista de duplicatas, inicialmente é feita uma busca para encontrar os registros que contêm esses IDs. Para isso, é usado o método isin(), que cria uma outra máscara lógica que pode ser usada no *DataFrame* para exibir os registros desejados, através do método .loc. O comando e os primeiros registros da busca são descritos à seguir:

```
df_limpo_1.loc[df_limpo_1['ID'].isin(ids_duplicados[0:3]), :].head(6)
```

Ao analisar os registros com IDs duplicados, nota-se que determinados IDs possuem valores válidos, mas existe um outro registro com o mesmo ID cujos valores são todos zero. Conclui-se que os registros com valores zero são inválidos.

Uma solução, é buscar os registros que possuem apenas valores zero, desconsiderando a coluna dos IDs. Os registros encontrados podem ser excluídos. Para isso, primeiro é criada uma máscara booleana para o *DataFrame*, que seleciona os registros com que considera a condição do valor igual a zero. Em seguida, é criada uma série que identifique os registros nos quais todos os atributos sejam iguais a zero, desconsiderando a coluna ID:

```
df_mascara_zeros = df_limpo_1 == 0 # Cria a máscara que seleciona os valores 0
serie_mascara_zeros = df_mascara_zeros.iloc[:,1:].all(axis=1) # Busca os registros cujos atributos sejam todos iguais a zero
sum(serie_mascara_zeros) # Retorna a quantidade de registros encontrados
```

O resultado é que 315 registros do conjunto de dados são compostos exclusivamente por valores 0. Para desconsiderar esses registros, é definido um outro *DataFrame* (df_limpo_2).

```
df_limpo_2 = df_limpo_1.loc[~serie_mascara_zeros,:].copy() # Usa a série e retorna o DataFrame sem os registros inválidos
df_limpo_2.shape # Retorna o tamanho do DataFrame após a operação
```

Atributos Categóricos

A coluna EDUCATION corresponde a um atributo do tipo categórico com as seguintes classes: 1 = 'pós-graduação'; 2 = 'graduação'; 3 = 'ensino médio'; 4 = 'outros'. O comando a seguir é útil para listar os valores do atributo e suas respectivas quantidades:

```
df_limpo_2['EDUCATION'].value_counts()
```

Ao analisar o atributo percebe-se que existem valores diferentes dos que constam na descrição do conjunto de dados (0, 5 e 6). Um modo de reduzir a quantidade de valores de um atributo consiste em reunir determinados valores em uma única categoria. No caso do exemplo, o valor 4 significa 'outros', sendo que as quantidades de valores 0, 5 e 6 serão considerados como da mesma categoria.

```
df_limpo_2['EDUCATION'].replace(to_replace=[0,5,6], value=4, inplace=True)
df_limpo_2['EDUCATION'].value_counts()
```

Codificação One-Hot (One-hot Encoding, OHE) é um modo alternativo para representação de dados categóricos. O método transforma atributos categóricos representados como números, cuja ordem não é relevante (não-ordenados), em um ou mais vetores binários esparsos. O objetivo é evitar que os modelos de aprendizado de máquina interpretem tais atributos como sendo numéricos.

Primeiro é definida a coluna que armazenará os valores categóricos ('EDUCATION_CAT').

```
df_limpo_2['EDUCATION_CAT'] = 'none' # Define um atributo 'EDUCATION_CAR' com valor none
df_limpo_2[['EDUCATION','EDUCATION_CAT']].head(10) # Mostra os 10 primeiros registros do atributo 'EDUCATION_CAT'
```

Em seguida é criado um dicionário que armazenará o mapeamento entre os valores numéricos do atributo e os valores textuais correspondentes às categorias:

```
map_categorias_ed = {
    1:'pos-graduacao',
    2:'graduacao',
    3:'ensino-medio',
    4:'outros'
}
```

Com o método .map() da biblioteca Pandas é possível estabelecer a relação entre os números e as strings d modo a preencher a nova coluna:

```
df_limpo_2['EDUCATION_CAT'] = df_limpo_2['EDUCATION'].map(map_categorias_ed)
df_limpo_2[['EDUCATION','EDUCATION_CAT']].head(10)
```

O método .get_dummies() Pandas realiza a codificação OHE da coluna desejada, gerando uma coluna para cada valor categórico do atributo:

```
edu_ohe = pd.get_dummies(df_limpo_2['EDUCATION_CAT'])
edu_ohe.head(10)
```

Por fim, é possível concatenar o novo *DataFrame* gerado ao *DataFrame* original:

```
df_ohe = pd.concat([df_limpo_2, edu_ohe], axis=1)
df_ohe[['EDUCATION_CAT', 'pos-graduacao', 'graduacao', 'ensino-medio', 'outros']].head(10)
```

Referências

- McKINNEY, Wes. "Python para análise de dados: Tratamento de dados com Pandas, NumPy e IPython". Editora Novatec, 2018.
- KLOSTERMAN, Stephen. "Projetos de Ciência de Dados com Python: Abordagem de estudo de caso para a criação de projetos de ciência de dados bem sucedidos usando Python, Pandas e Scikit-Learn. Editora Novatec, 1ª Edição, 2020.
- GÉRON, Aurélien. "Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow". O'Reilly Media, Inc., 2022.
- Biblioteca Pandas Python
- Biblioteca Pandas-Profile Python
- ISMIGUZEL, Idil. "Imputing Missing Data with Simple and Advanced Techniques" Acesso em 04/02/2023, Disponível em: https://towardsdatascience.com/imputing-missing-data-with-simple-and-advanced-techniques-f5c7b157fb87
- BROWNIEE, Jason. "How to Use StandardScaler and MinMaxScaler Transforms in Python". Acesso em 04/02/2023, Disponível em https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/
- MAIA, Nayane. "How to create and interactive map of Brazil using Plotly.Express-Geojson in Python". Acesso em 04/02/2023, Disponível em https://python.plainenglish.io/how-to-create-a-interative-map-using-plotly-express-geojson-to-brazil-in-python-fb5527ae38fc