Capítulo 2

COVID-19: Aquisição, tratamento e visualizações interativas de dados do Ministério da Saúde

Alexandre R. C. Ramos, Jonnison L. Ferreira, Moisés Laurence de F. Lima Junior, Aristofánes Corrêa Silva

Abstract

Given the COVID-19 pandemic, declared on March 11, there was an exponential increase in the available data. The data science area, in this context, can produce accurate and objective computational solutions to reduce the impact of the pandemic on some aspects of our life in society. In this perspective, this chapter presents the workflow for building interactive visualizations of COVID-19 in Brazil, involving the acquisition and treatment of data obtained from the official panel of the Ministry of Health. The techniques and tools presented enabled direct and accurate visualizations, which can help decision making by the public authorities and guarantee the communication of pandemic information in Brazilian states and regions.

Resumo

Diante da pandemia da COVID-19, declarada em 11 de março, houve um aumento exponencial na disponibilização de dados relacionados ao tema. O campo da ciência de dados, neste contexto, pode produzir soluções computacionais tempestivas e diretas para contribuir com a redução do impacto da pandemia em nossa sociedade. Nesta seara, este capítulo apresenta o fluxo de trabalho para construção de visualizações interativas da COVID-19 no Brasil, envolvendo a aquisição e tratamento de dados obtidos no painel oficial do Ministério da Saúde. As técnicas e ferramentas apresentadas possibilitaram visualizações diretas e precisas, que podem auxiliar a tomada de decisão pelo poder público, bem como garantir a comunicação das informações da pandemia nos estados e regiões brasileiras.

2.1. Introdução

O surto de COVID-19 foi relatado pela primeira vez em Wuhan (China) e se espalhou para mais de 50 países de forma assustadoramente rápida, assim, a Organização Mundial

da Saúde (OMS) declarou a COVID-19 como Emergência de Saúde Pública de Interesse Internacional (PHEIC) em 30 de Janeiro de 2020. Diante do aumento exponencial no número de casos registrados, o que fatalmente acarretaria em um colapso de sistemas de saúde pelo mundo, em 11 de março a infecção causada pela SARS-CoV-2 foi caracterizada como uma pandemia. No Brasil, o crescimento acelerado da doença também apresentou destaque e preocupação, pois rapidamente a situação evoluiu de casos importados (primeiro caso registrado em 26 de fevereiro) à transmissão comunitária (confirmada em 20 de março) [Datasus 2020].

Com o estado de pandemia declarado pela OMS, houve um aumento exponencial de dados relacionados ao tema sendo disponibilizados por diferentes órgãos oficiais [Datasus 2020, Government 2020]. Neste cenário, as técnicas e ferramentas para análise e visualização destes dados mostram-se fundamentais para o planejamento de ações governamentais contundentes e tempestivas, como também constituem ferramentas para a comunicação efetiva da população sobre os problemas acarretados pela pandemia.

Em [Doyle 2020] é destacada a necessidade crítica de dados epidemiológicos oportunos, precisos e acessíveis no acompanhamento da COVID-19. Dados de casos com detalhamento geográfico são particularmente valiosos durante surtos, mas raramente disponibilizados em tempo real e/ou apresentados de forma simples. Desta maneira, se torna fundamental e necessário o desenvolvimento de ferramentas para visualizações interativas de dados, possibilitando a comunicação e contextualização das informações.

Esse capítulo discorre sobre os aspectos teóricos e práticos de etapas relevantes no campo da ciência de dados aplicada à COVID-19, bem como suas possíveis contribuições no enfrentamento da pandemia. Assim, apresentaremos técnicas e ferramentas utilizadas para a aquisição e tratamento de dados (Seção 2.2) e na construção de visualizações interativas (Seção 2.3), tendo como base o conjunto de dados fornecidos pelo portal da COVID-19 do Ministério da Saúde.

2.2. Aquisição e Tratamento de Dados

2.2.1. Aquisição de Dados

Esta seção apresenta um fluxo de trabalho para a aquisição e tratamento de dados do Ministério da Saúde sobre o Covid-19. Essa etapa consiste em acessar as fontes de dados. No nosso caso, os dados estão disponíveis no sítio[1] do ministério da saude dedicado ao acompanhando da disseminação do vírus no território nacional.

A Figura 2.1 aprensenta a página inicial do site do ministério da saúde dedicado ao monitoramento do Covid-19. Onde podemos verificar alguns gráficos e interações. Para obter os dados em um formato ideal a ser processado, devemos clicar no botão ”Arquivo CSV”, o qual destacamos na Figura 2.1 com uma caixa vermelha. Ao clicar no botão somos então redirecionados para o *download* de um arquivo *CSV (Comma Separeted Values)* com as informações do Covid-19 organizadas por data e estado.

Realizar essa tarefa manualmente torna o processo aquisição dos dados muito lento, e deixando dependente de uma iteração manual, onde o administrador do sistema

[1] https://covid.saude.gov.br/

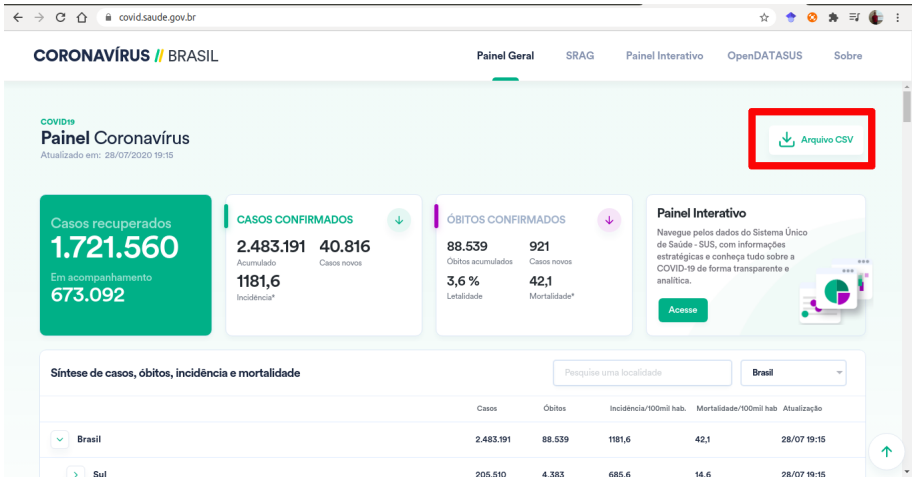

Figura 2.1. Página inicial do site do ministério da saúde dedicado ao monitoramento do Covid-19

deve realizar o processo manualmente. Existem diversas técnicas que permitem automatizar esse processo, esse tipo de aplicação é chamada de *Web Scraping*. Mitchell (2019) define *Web Scraping* como um conjunto de técnicas que permitem a mineração de dados de diversas fontes na *web* de maneira automatizada, onde determinados *scripts* acessam o conteúdo do site e extraem as informações importantes para o contexto analisado.

Esse processo tem funcionamento baseado em requisições *HTTP* automáticas ao sítios onde as informações relevantes são armazenadas, após a requisição o *HTML* é processado e a informação relevante é extraída. Existem diversas bibliotecas que oferecem uma *API* amigável para a execução dessas requisições, como o *python-requests* para execução da requisição [Reitz 2020] e *BeatifulSoap*[2], que são alternativas de código aberto bem populares.

O site do Ministério da saúde altera diariamente o a *URL* do arquivo *CSV*, além disso o link para *download* do arquivo é gerado automaticamente por meio do controle de eventos de interação com o usuário através da execução de *JavaScript*. Portanto além de automatizar a requisição é necessário realizar o *scraping* do *JavaScript*, permitindo então a simulação da execução da ação de clicar no botão de acesso ao arquivo, para então capturar o link do arquivo.

Utilizaremos o *Selenium*[3] para realizar esse *Scraping de Javascript*. O *Selenium* é um *framework* de código aberto que permite a execução automatizada de diversas tarefas pelo browser, proposto inicialmente como ferramenta de testes automatizados, também é muito utilizado para essa captura de informações na *web* [Mitchell 2019].

2.2.1.1. Utilizando o *Selenium* para capturar os dados

Primeiro passo para utilização do *Selenium* juntamente com o *Python* é a instalação da biblioteca, para isso é utilizado o Código Fonte 2.1.

[2]http://www.crummy.com/software/BeautifulSoup/bs4/
[3]http://docs.seleniumhq.org/

```
1 pip install selenium
```

Código Fonte 2.1. Instalando o *Selenium*.

Após a instalação, iniciaremos nosso *script* de *download* automático do arquivo *CSV*, o Código Fonte 2.2, responsável adicionar as bibliotecas que utilizaremos no *script*. A primeira linha é responsável por importar o *webdriver*. O *webdriver* é o navegador onde o *Selenium* irá executar a requisição automática, o nosso caso utilizaremos o navegador *Mozilla Firefox*, além deste o *Selenium* suporta outros navegadores, como o *Google Chrome* e *PhantomJS*. A segunda linha importa da classe *Python* que utilizaremos para configurar o firefox. Por fim importaremos as bibliotecas *glob* e *os* que utilizaremos para mantermos apenas a versão mais atual do *CSV* disponibilizado pelo ministério da saúde.

```
1 from selenium import webdriver
2 from selenium.webdriver.firefox.options import Options
3 import glob,os
```

Código Fonte 2.2. Importando bibliotecas necessárias.

O passo seguinte será apagar as versões anteriores dos arquivos e configurar o navegador para *download* do arquivo da atualizado do sítio do ministério da saúde. O Código Fonte 2.3 é o código que utilizaremos para realizar essa configuração. A linha 1 e 2 definem onde ficaram os arquivos do *CSV* original e o *CSV* após o tratamento para evitar inconsistências, respectivamente. As linhas 4 e 5 percorrem o diretório de *download* e apagam as versões anteriores dos arquivos a serem baixados.

O código das linhas 7 a 11 servem para configurar o navegador para realizar o *download* de qualquer arquivo selecionado pelo *script* e salvar no diretório correto sem a necessidade de confirmação do usuário caso o arquivo seja *CSV*. As linhas 13 e 14 configuram o navegador que o mesmo seja executado em plano fundo, sem a necessidade de abrir a interface gráfica.

```
1  download_path = '/tmp/'
2  output_path = '/webapps/covid/static/'
3
4  for file in glob.glob(download_path+"*.csv"):
5    os.remove(file)
6
7  profile = webdriver.FirefoxProfile()
8  profile.set_preference("browser.download.folderList", 2)
9  profile.set_preference("browser.download.manager.showWhenStarting",
       False)
10 profile.set_preference("browser.download.dir", download_path)
11 profile.set_preference("browser.helperApps.neverAsk.saveToDisk","text/
       csv,application/csv,text/plan,text/comma-separated-values")
12
13 options = Options()
14 options.headless = True
```

Código Fonte 2.3. Apagando versões anteriores e configurando o navegador.

Com as configurações realizadas, basta abrir o navegador e acessar o site, as linhas 1 e 2 do Código Fonte 2.4 abrem o navegador com as configurações definidas e

acessa o sítio do ministério da saúde. Após acessar o site precisamos encontrar o botão de *download*, para isso devemos identificá-lo de maneira única na página, ao analisar o sitio percebemos que ele é o botão a possuir a classe "*btn-outline*", com isso, nas linhas 4 e 5 encontramos o botão e realizamos *download*. A linha 7 é utilizada para que o navegador seja encerrado após o download do arquivo. Com isso encerramos o *script* de *download* e podemos tratar os dados.

```
1 driver=webdriver.Firefox(firefox_profile=profile,options=options)
2 driver.get("https://covid.saude.gov.br/")
3
4 btn = driver.find_element_by_class_name("btn-outline")
5 btn.click()
6
7 driver.close()
```

Código Fonte 2.4. Abrindo o navegador e acessando o sítio.

2.2.2. Tratamento de Dados

Em aplicações onde buscamos dados de diversas fontes, é necessário o tratamento dos dados, para que não hajam inconsistências [Grus 2019]. Mesmo utilizando uma única fonte de dados precisaremos fazer alguns tratamentos nos dados, para isso utilizaremos a biblioteca pandas[4].

O Pandas fornece uma sério de estruturas e funções de dados avançadas, projetadas para torna o trabalho com dados estruturados mais rápido e simples, fornecendo um ambiente de análise de dados poderoso e produtivo [McKinney 2012].

O arquivo fornecido pelo ministério da saúde possui algumas inconsistências, como datas em formatos diferentes e informações que mudam de um dia para o outro. Para resolver esses problemas utilizaremos a biblioteca *pandas*.

O Código Fonte 2.5 mostra o tratamento dos dados, na linha 1 importamos as bibliotecas necessárias, usaremos *glob* para encontrar o arquivo no diretório de download, como pode ser visto na linha 3. Na linha 4 realizaremos a leitura do arquivo.

As linhas 5 e 6 configuramos as inconsistências que podem ocorrer no campo que define a data da coleta e convertemos os dados da coluna para o tipo *datetime64*. Na linha salvamos o arquivo *CSV* processado no local definido no Código Fonte 2.3, selecionando apenas os campos que queremos e formatando a data corretamente, para um melhor tratamento na interface *web*.

```
1 import pandas,glob
2
3 csv_name = glob.glob(download_path+"*.csv")[0]
4 pd_df = pd.read_csv(csv_name,sep=";")
5 pd_df = pd_df.rename(columns=lambda x: "data" if "data" in x.lower()
      else x)
6 pd_df['data'] = pd_df['data'].astype('datetime64[ns]')
7 pd_df.to_csv(output_path+"minSaude.csv",header=['regiao','estado','data
      ','casosNovos','casosAcumulados','obitosNovos','obitosAcumulados'],
      index=False,date_format='%d/%m/%Y')
```

[4]https://pandas.pydata.org/

Código Fonte 2.5. Tratando os dados com o *pandas*.

2.3. Visualização de Dados

Esta seção apresenta um fluxo de trabalho para a construção de visualizações da *COVID-19* após a aquisição e tratamento de dados do Ministério da Saúde. Assim, apresentaremos inicialmente as visualizações propostas e seu contexto (Subseção 2.3.1) e discutiremos a codificação do projeto e a utilização de um conjunto de bibliotecas relacionadas à construção de gráficos e mapas (Subseção 2.3.2). Vale ressaltar: para a devida compreensão e possibilidade de adaptações nas visualizações produzidas nesse trabalho, são necessários conhecimentos introdutórios relacionados à programação Web (HTML, CSS, *Javascript*, DOM, dentre outros).

O fluxo de trabalho e as informações discutidas nessa seção partem da estruturação dos dados fornecidos pelo ministério da saúde em um arquivo de dados tabelados (CSV), cuja construção é descrita na Seção 2.2. Este arquivo[5] é constituído pelos dados da COVID-19 no Brasil organizados por região, estado, data, número de novos casos, número de casos acumulados, número de novos óbitos e número de óbitos acumulados.

2.3.1. Apresentação

As visualizações apresentadas neste trabalho tem como objetivo tornar simples e interativas as informações relacionadas à COVID-19 presentes no painel do Ministério da Saúde[6]. Portanto, nos concentramos em questões simples e diretas que poderão ser respondidas rapidamente na leitura das visualizações propostas. Por exemplo:

1. ”Qual o número de casos confirmados e de óbitos do meu estado?”
2. ”Como está a situação do meu estado em relação à sua região ou ao Brasil?”
3. ”Como aconteceu a evolução temporal de novos casos e óbitos em meu estado ou Região?”

As Figuras 2.2 e 2.3 ilustram as visualizações propostas, formadas por um painel, com mapas temáticos dos estados e regiões brasileiras (Figura 2.2), e um *DashBoard* interativos (Figura 2.3). Ao refletir sobre as questões apontadas inicialmente, podemos observar que as variáveis região, estado, casos novos e acumulados, óbitos novos e acumulados e data se repetem em diferentes combinações. Neste contexto se insere a interatividade proposta em nossas visualizações, buscando possibilitar ao público alvo uma navegação simples com dados consistentes e diretos.

No painel de mapas temáticos três controles foram possibilitados ao usuário: estado ou região; incidência ou letalidade e data. Além disso as informações detalhadas de cada estado podem ser acessadas pela legenda do canto superior direito, quando o

[5]Disponível aqui: `https://drive.google.com/file/d/1cc6T4fLSixunlt-AcndJY7q6DP7lD6bV/view?usp=sharing`

[6]https://covid.saude.gov.br/

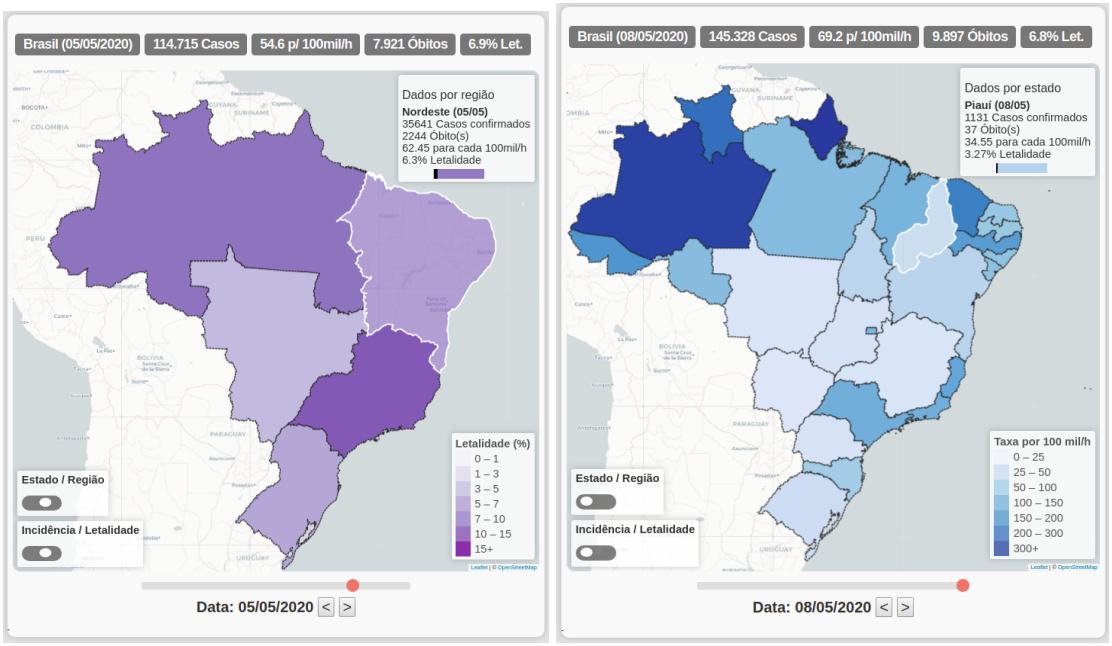

Figura 2.2. Mapas interativos do Coronavírus no Brasil. (A) Letalidade por Região e (B) Incidência por estado. Também disponível em vídeo, no link: `https://youtu.be/rOKTCQCueCc`

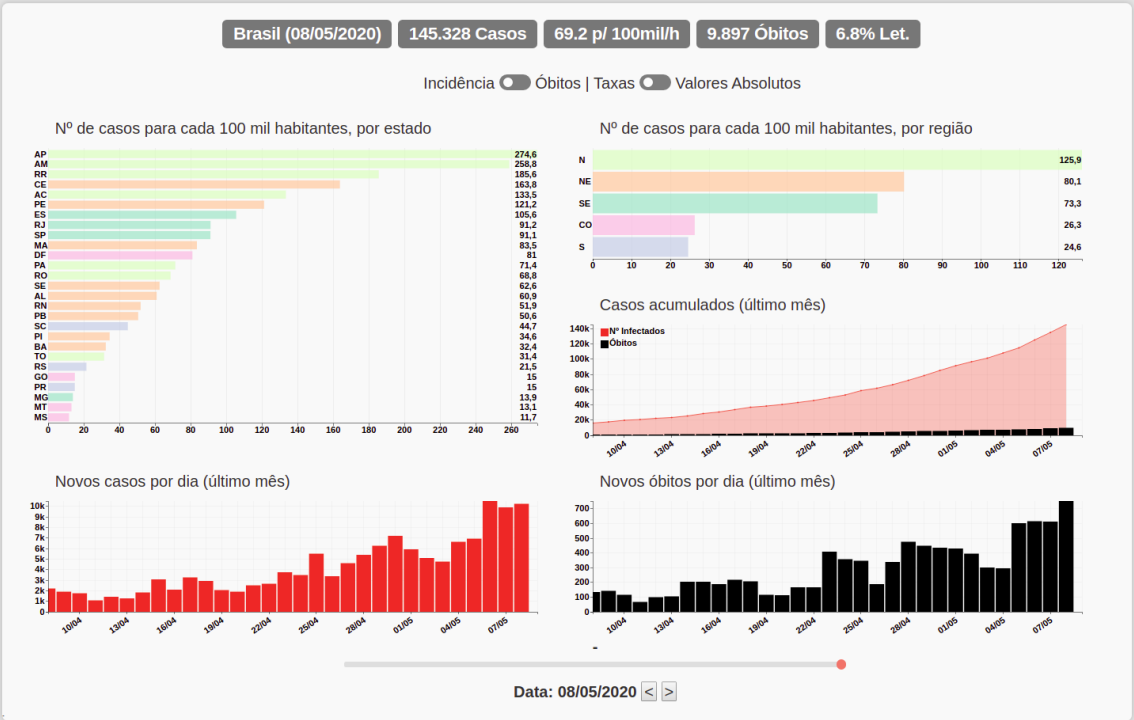

Figura 2.3. DashBoard Interativo do Coronavírus no Brasil. Também disponível em vídeo, no link: `https://youtu.be/yBpQFCCEFuA`

mouse se encontra sobre o respectivo limite geográfico. Já no *Dashboard* os próprios gráficos estão interligados como filtros. É possível observar a evolução temporal das variáveis de uma região ou estado clicando em sua respectiva barra horizontal. Além disso, também foram possibilitados três controles: incidência ou óbitos; taxas ou valores absolutos e data. Assim, a interatividade proposta possibilita múltiplas combinações e evita uma visualização extensa e cansativa, reduzindo também o espaço em tela necessário à comunicação.

Para definição dos elementos visuais presentes nestas visualizações, tais como a seleção de escala de cores, tipos de gráficos, conjunto de dados, experiência do usuário, dentre outros, utilizamos como base as informações apresentadas nos livros *Storytelling with Data: A Data Visualization Guide for Business Professionals* [Knaflic 2015] e *The Truthful Art: Data, Charts, and Maps for Communication* [Cairo 2015].

A partir da disponibilização de visualizações como estas em um ambiente Web, um grande número de pessoas podem ter acesso e navegar por informações da pandemia em seu estado ou região, reduzindo a gigantesca carga de informação disponibilizada em outros veículos e tratando do assunto de maneira direta, simples e interativa. Projetos como este podem, portanto, gerar contribuições significativas em diferentes contextos e problemáticas. Ademais, o domínio de técnicas de visualização, como as utilizadas neste trabalho, capacitam profissionais para a construção de comunicações robustas e eficazes.

2.3.2. Codificação e Bibliotecas utilizadas

Esta subseção apresenta um conjunto de bibliotecas e os principais trechos de códigos utilizados na construção das visualizações elaboradas neste trabalho. Com o desenvolvimento em uma plataforma Web, utilizando a linguagem *Javascript*, destacamos a utilização das bibliotecas D3js (Subsubseção 2.3.2.1), Leaflet (Subsubseção 2.3.2.2), DC e Crossfilter (Subsubseção 2.3.2.3). Ademais, a Subsubseção 2.3.2.4 demonstra os principais trechos de código desenvolvidos.

2.3.2.1. D3js

D3js[7] é uma biblioteca JavaScript que utiliza os padrões HTLM, SVG e CSS para gerar visualizações de dados interativas e dinâmicas. Sua ênfase nos padrões da web possibilita a utilização de todos os recursos dos navegadores modernos sem a necessidade de se vincular a estruturas proprietárias [Meeks 2015].

A D3js permite a manipulação de documentos com base em dados. Possui agilidade, comportando grandes conjuntos de dados e funcionamento dinâmico para interação e animação, ademais seu estilo funcional possibilita a reutilização de códigos através de uma coleção diversificada de módulos oficiais e desenvolvidos pela comunidade[8]. Neste trabalho, a D3js é responsável pela leitura da base de dados e pela manipulação de elementos do DOM, como demonstrado na Subsubseção 2.3.2.4.

[7] https://d3js.org/

[8] Disponíveis em https://observablehq.com/@d3/gallery

2.3.2.2. Leaflet

Leaflet[9] é uma biblioteca *JavaScript* de código aberto utilizada para criar mapas interativos, como ilustra a Figura 2.4. É projetada para funcionar de maneira eficiente nas principais plataformas desktop e mobiles e pode ser estendida através de plugins ou apresentar uma API de fácil utilização e código-fonte simples e legível, com diversos exemplos e tutoriais oficiais disponibilizados.

Neste trabalho, a *Leaflet* foi fundamental para a construção do painel com mapas interativos, ilustrado na Figura 2.2, o que pode ser observado nos respectivos códigos fonte comentados (Subsubseção 2.3.2.4). Além disso, quatro tutoriais oficiais são recomendados para a devida compreensão deste painel: *Leaflet Quick Start Guide*[10]; *Using GeoJSON with Leaflet*[11]; *Interactive Choropleth Map* [12]; E *Working with map panes*[13].

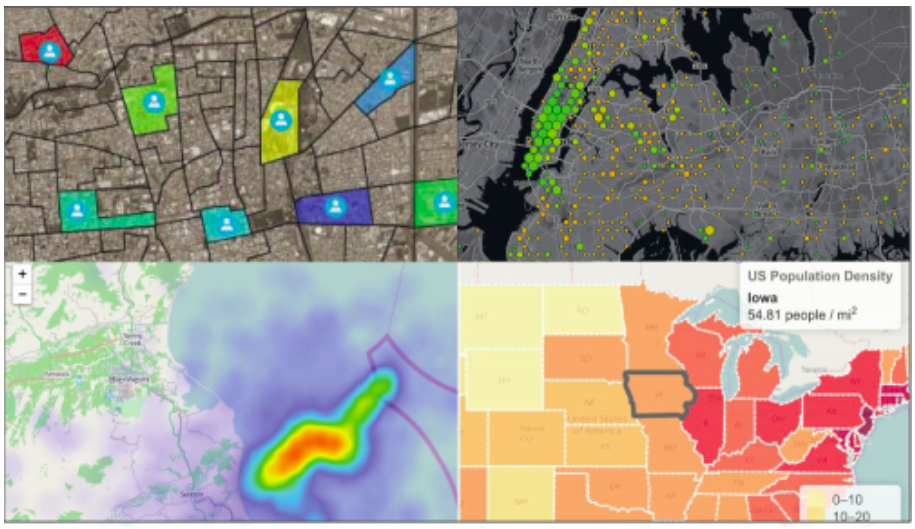

Figura 2.4. Exemplos de mapas construídos a partir da leaflet. Fonte: https://leafletjs.com/examples.html

2.3.2.3. DCjs e *Crossfilter*

Especificamente para a construção de *Dashboards* na Web, duas bibliotecas utilizadas se destacam: DCjs[14] e *Crossfilter*[15]. DCjs consiste em uma biblioteca *JavaScript* utilizada para a renderização de gráficos, com suporte nativo *Crossfilter*, que, por sua vez, é utilizada para análise e exploração extremamente rápida de grandes conjuntos de dados. Juntas, estas bibliotecas possibilitam a construção simplificada e eficaz de *Dashboards* interativos, como ilustrado na Figura 2.5.

[9] https://leafletjs.com/
[10] https://leafletjs.com/examples/quick-start/
[11] https://leafletjs.com/examples/geojson/
[12] https://leafletjs.com/examples/choropleth/
[13] https://leafletjs.com/examples/map-panes/
[14] https://dc-js.github.io/dc.js/
[15] https://square.github.io/crossfilter/

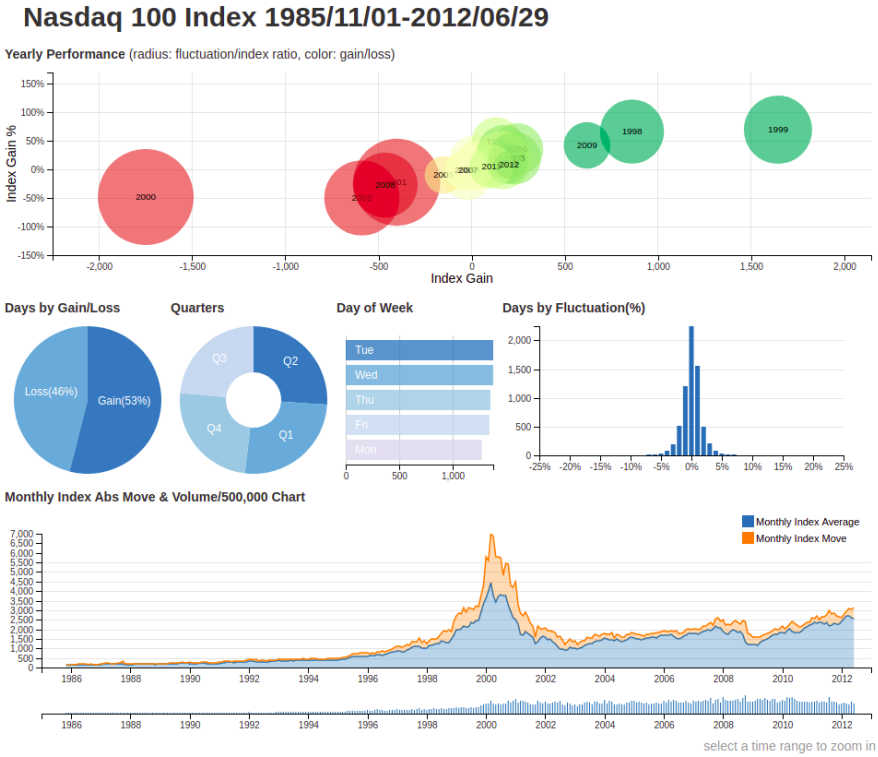

Figura 2.5. Exemplos de *DashBoard*, com diferentes tipos de gráficos, construído a partir da DCjs. Disponível em https://dc-js.github.io/dc.js/

Destaca-se, inicialmente, que a compreensão dos conceitos de dimensão e agrupamento *Crossfilter* são fundamentais [16]. Neste trabalho, o *Dashboard* interativo ilustrado na Figura 2.3 tem como base quatro exemplos DCjs oficiais simples: *Ordinal Line*[17]; *Ordinal Bar*[18]; *Row*[19]; E *Composite*[20]. As adaptações e alterações necessárias, como a formatação de números, rotação de legendas e manipulação de outros elementos visuais, foram possibilitadas a partir das suas respectivas documentações[21] e podem ser observadas nos códigos fonte (Subsubseção 2.3.2.4).

2.3.2.4. Principais trechos de códigos

Nesta seção apresentaremos os principais trechos de código relacionados à construção das visualizações produzidas. Considerando que a estilização (CSS) e marcação (HTML) de elementos não são temáticas centrais deste trabalho, bem como a demonstração do projeto completo seria extensiva e fora de contexto, destacamos três trechos de código fundamentais, com comentários explicativos, para possibilitar a compreensão e construção

[16] Um breve e conciso tutorial está disponível em: `https://www.tutorialspoint.com/dcjs/dcjs_introduction_to_crossfilter.htm`

[17] https://dc-js.github.io/dc.js/examples/ordinal-line.html

[18] https://dc-js.github.io/dc.js/examples/ordinal-bar.html

[19] https://dc-js.github.io/dc.js/examples/row.html

[20] https://dc-js.github.io/dc.js/examples/composite.html

[21] Disponíveis em: https://dc-js.github.io/dc.js/docs/html/

das visualizações apresentadas.

O Código Fonte 2.6 demonstra a leitura e manipulação dos dados da COVID-19 no Brasil utilizando as bibliotecas *D3js* e *Crossfilter*. Inicialmente os dados são tratados e armazenados em variáveis com a *D3js* (entre as linhas 1 e 11) e, em seguida, duas dimensões com chaves simples e composta (linhas 15 e 37) e grupos (linhas 20 e 39) *Crossfilter* são definidos e manipulados. Este código ilustra a leitura e acesso aos dados no ambiente de desenvolvimento Web com as respectivas bibliotecas.

```
1  d3.csv("../minSaude.csv", function(data){
2      dtgFormat = d3.time.format("%d/%m/%Y");
3      data.forEach(function(d) {
4          d.regiao = d.regiao;
5          d.uf = d.estado;
6          d.data = dtgFormat.parse(d.data);
7          d.casosNovos = +d.casosNovos;
8          d.casosAcumulados = +d.casosAcumulados;
9          d.obitosNovos = +d.obitosNovos;
10         d.obitosAcumulados = +d.obitosAcumulados;
11     });
12     //Leitura de dados pela CrossFilter:
13     var facts = crossfilter(data);
14     // Dimensao CrossFilter com chave composta (Estado + Data):
15     var Data_UF = facts.dimension(
16         function(d){
17             return 'UF:'+d.uf +', data:'+d.data
18         }),
19     // Agrupando o numero de casos por data e estado:
20     groupCasos_DataUF =
21         Data_UF.group().reduceSum(function(d){return d.casosAcumulados
       ;});
22         // Acessando elementos do grupo CrossFilter:
23     groupCasos_DataUF.all()
24             .forEach(function(d){
25                     console.log(d.key) // UF: x, data: y;
26                     console.log(d.value) // Total de casos acumulados.
27             });
28         // Acessando um elemento a partir de uma chave ('SP',
       '10/04/2020'):
29         var key = 'UF:'+'SP'+', data:'+dtgFormat.parse('10/04/2020'),
30         resposta = groupCasos_DataUF.all()
31         .filter(
32             function(i){
33                 return i.key == key
34         })[0].value;
35         console.log('total de casos de SP em 10/04/2020: '+resposta);
36     // Dimensao CrossFilter com chave simples (Regiao):
37         var Regiao = facts.dimension(function(d){return d.regiao;});
38     // Agrupando obitos por regiao (Todos os registros de obitos novos
       por regiao serao somados):
39         var groupObitos = Regiao.group()
40         .reduceSum(function(d){
41                     return d.obitosNovos;
42         });
43     // Acessando um elemento a partir de uma chave ('Nordeste'):
```

```
44          var resposta = groupObitos.all().filter(function(i){ return i.
    key == 'Nordeste' })[0].value;
45          console.log(resposta);
46
47  }
```

Código Fonte 2.6. Leitura e manipulação dos dados com D3js e *Crossfilter*

O Código Fonte 2.7 ilustra a construção de um mapa temático dos casos de COVID-19 no Brasil em 10 de maio de 2020 com *Leaflet*. Neste código, o painel é inicializado (linhas 1 a 7) com mapas *OpenStreetMap* [22]. Em seguida, definimos uma data de referência (linha 9) e uma escala linear (linhas 10 a 12) que será utilizada para mapear e colorir os estados a partir do número de casos [23]. Entre as linhas 14 e 19 acrescentamos os limites dos estados [24], com a chamada de duas funções para estilização e iteração de cada polígono (Ou limite geográfico de cada estado). Na função **estilização** (linhas 21-32), acessamos o número de casos por estado como já demonstrado no Código Fonte 2.6 e atribuímos a respectiva cor utilizando o mapa linear previamente definido. Já a função **ParaCadaEstado** (linhas 33-38) acrescenta o tratamento de eventos *Leaflet* [25] aos polígonos, neste caso utilizamos o evento *click* para informar ao usuário o valor exato do número de casos de um estado (linhas 40-50). Por fim, elaboramos um *Leaflet Control* [26] para representar a legenda do mapa (linhas 52-76).

```
1     let centro = [-14, -54],zoom = 4,
2     Mapa = L.map('divMapa', { zoomControl:false }).setView(centro, zoom
      );
3
4     L.tileLayer(
5     'https://cartodb-basemaps-{s}.global.ssl.fastly.net/light_all/{z}/{
      x}/{y}.png',
6     {maxZoom: 18, attribution: '&copy; <a href="http://www.
      openstreetmap.org/copyright">OpenStreetMap</a>'}
7     ).addTo(Mapa);
8
9     let date = dtgFormat.parse('10/05/2020'),
10    escalaDeCores = d3.scale.linear()
11    .domain([0,1000,5000,10000,15000,25000,40000])
12    .range(['#edf8fb','#ccece6','#99d8c9','#66c2a4','#41ae76','#238b45'
      ,'#005824']);
13
14    L.geoJson(
15        Estados, {
16            style: estilizacao,
17            onEachFeature: ParaCadaEstado,
```

[22] https://www.openstreetmap.org/

[23] Escala de cores disponível em: https://colorbrewer2.org/#type=sequential&scheme=BuGn&n=7

[24] **Estados** é uma variável previamente definida, contendo os limites geográficos dos estados brasileiros em *GeoJson*, elaborada a partir do projeto Geodata BR - Brasil, disponível em: https://github.com/tbrugz/geodata-br

[25] Outros eventos são descritos na documentação oficial, disponível em: https://leafletjs.com/reference-1.6.0.html#map-event

[26] Conforme descrito em: https://leafletjs.com/reference-1.6.0.html#control

```
18          }
19      ).addTo(Mapa);
20
21      function estilizacao(feature){
22          var key = 'UF:'+feature.properties.UF+', data:'+date,
23          Ncasos = groupCasos_DataUF.all().filter(function(i){ return i.
    key == key })[0].value;
24          return {
25              weight: 2,
26              opacity: 1,
27              color: '#242424',
28              dashArray: '3',
29              fillOpacity: 1,
30              fillColor: escalaDeCores(Ncasos)
31          };
32      }
33      function ParaCadaEstado(feature, layer) {
34          layer._leaflet_id = feature.properties.UF;
35              layer.on({
36                  click: clickEstado
37                  // Outros Eventos_Leaflet: ...,
38          });}
39
40      function clickEstado(e) {
41          let layer =  e.target;
42          var cod = layer.feature.properties.UF;
43          var key = 'UF:'+cod+', data:'+date,
44          Ncasos = groupCasos_DataUF.all()
45              .filter(
46                  function(i){
47                      return i.key == key
48                  })[0].value;
49          alert('Estado: '+cod+';\nN de Casos em '+date+': '+Ncasos);
50      }
51
52      let legenda = L.control({position: 'bottomright'});
53
54      legenda.onAdd = function (map) {
55        var title = 'Num de Casos';
56          var format = d3.format("s");
57          var cores = escalaDeCores.range();
58          let div = L.DomUtil.create('div', 'legenda'),
59          labels = [],
60          n = cores.length,
61          from, to;
62          labels.push(title);
63          for (let i = 0; i < n; i++) {
64              let c = cores[i];
65              let fromto = escalaDeCores.domain();
66              var v1 = format(d3.round(fromto[i],1)),
67              v2 = d3.round(fromto[i+1],1);
68              labels.push(
69              '<i style="background:' + cores[i] + '"></i> ' +
70              v1 + (v2 ? ' &ndash; ' + format(v2) : '+'));
71          }
```

```
72          div.innerHTML = labels.join('<br>');
73          return div;
74      }
75
76      legenda.addTo(Mapa);
```

Código Fonte 2.7. Construção de um mapa temático com o número de casos da COVID19 por estado em 10/05/2020.

O Código Fonte 2.8 demonstra a construção de dois gráficos interativos e interconectados com *D3*, *DC* e *Crossfilter*. O gráfico 1 representa o número total de casos por estado em barras horizontais e o gráfico 2 o número de novos casos por dia em barras verticais. Inicialmente definimos as dimensões e grupos *Crossfilter* que serão utilizados nestes gráficos (linhas 1 a 8), que foram inicializados e atribuídos às respectivas *divs HTML* (linhas 13 e 14). A construção destes gráficos, presente nas linhas 16 a 33 e 34 a 47, seguem os exemplos e documentações descritos na Subsubseção 2.3.2.3, onde é possível compreender cada elemento estabelecido. Destacam-se, portanto, as adaptações realizadas para o público brasileiro: Formatação dos números e data entre as linhas 54 e 72. Além disso, entre as linhas 49 e 52 realizamos um filtro na dimensão data para que apenas os valores até o dia 10 de maio sejam considerados e entre as linhas 74 e 78 demonstramos como rotacionar a legenda para o eixo X do gráfico 2. Com este trecho simples de código, as bibliotecas *DC* e *Crossfilter* se encarregam dos filtros e demais interações entre gráficos[27].

```
1       // Dimensao e grupo para o grafico 1:
2       var dim_UF = facts.dimension(function(d){return d.uf;}),
3       Casos_UF = dim_UF.group()
4           .reduceSum( function(d){return d.casosNovos;});
5       // Dimensao e grupo para o grafico 2:
6       var dim_Data =  facts.dimension(function(d){return d.data;}),
7       Casos_Data = dim_Data.group()
8           .reduceSum(function(d){return d.casosNovos;});
9       // Janela temporal para grafico 2:
10      var maxDate = dtgFormat.parse("10/05/2020"),
11      minDate = d3.time.day.offset(maxDate, -20);
12      // Inicializacao dos graficos:
13      var grafico1 = new dc.rowChart("#divGrafico1");
14      var grafico2 = new dc.barChart("#divGrafico2");
15      // Contrucao dos graficos:
16      grafico1
17          .width(900)
18          .height(800)
19            .margins({left: 100, top: 10, right: 50, bottom: 60})
20          .renderLabel(true)
21              .renderTitleLabel(true)
22          .labelOffsetX(-35)
23          .elasticX(true)
24          .dimension(dim_UF)
25              .group(Casos_UF)
26          .title(function(d){
27            if(d.value == 0)
```

[27]Outras informações detalhadas podem ser encontradas em https://github.com/square/crossfilter/wiki/API-Reference

```
                return '-';
            return d.value.toLocaleString("pt-BR");
              })
              .colorAccessor(function (d){return d.value;})
              .colors(function(d){
                  return escalaDeCores(d);});
    grafico2
        .width(900)
        .height(500)
        .elasticY(true)
            .margins({left: 60, top: 30, right: 80, bottom: 60})
            .x(d3.time.scale().domain([minDate, maxDate]))
        .xUnits(d3.time.days)
        .centerBar(true)
        .brushOn(false)
        .renderHorizontalGridLines(true)
            .renderVerticalGridLines(true)
        .colors(['#e34a33'])
        .dimension(dim_Data)
        .group(Casos_Data);
    // Exemplo de filtro com dimensao crossfilter
    dim_Data.filter(function(d){
            if(d<=dtgFormat.parse("10/05/2020"))
            return d;
        });
    // Formatacao de numeros e data no grafico 2:
    var formatDay = d3.time.format("%d"),
    formatMonth = d3.time.format("%m");
    grafico2
        .title(function(d){
            var dia   = formatDay(d.key),
            mes = formatMonth(d.key);
            return ('('+dia+'/'+mes+'): '+d.value+'');
        })
        .xAxis()
            .ticks(d3.time.days, 3)
            .tickFormat(function(d){
                var dia = formatDay(d),
                mes = formatMonth(d);
                return (dia+'/'+mes);
            })
        .yAxis()
                .tickFormat(function(d){
                    return d.toLocaleString("pt-BR");
                });
    // Rotacao da legenda inferior - Eixo X - no grafico 2:
    grafico2
        .on('renderlet', function (chart) {
            chart.selectAll('g.x text')
            .attr('transform', 'translate(-20,10) rotate(-35)')
        });
    // Renderizacao dos graficos:
    dc.renderAll();
```

Código Fonte 2.8. Construção de dois gráficos interligados e interativos com *D3 DC* e *Crossfilter*.

Os três trechos de código apresentados são subsequentes e inter-relacionados, exemplificando a utilização de cada biblioteca no fluxo de trabalho para construção de visualizações interativas da COVID-19. Estes trechos, associados aos respectivos elementos HTML e sua estilização[28], representam versões simplificadas das visualizações construídas neste trabalho, seus resultados são ilustrados na Figura 2.6.

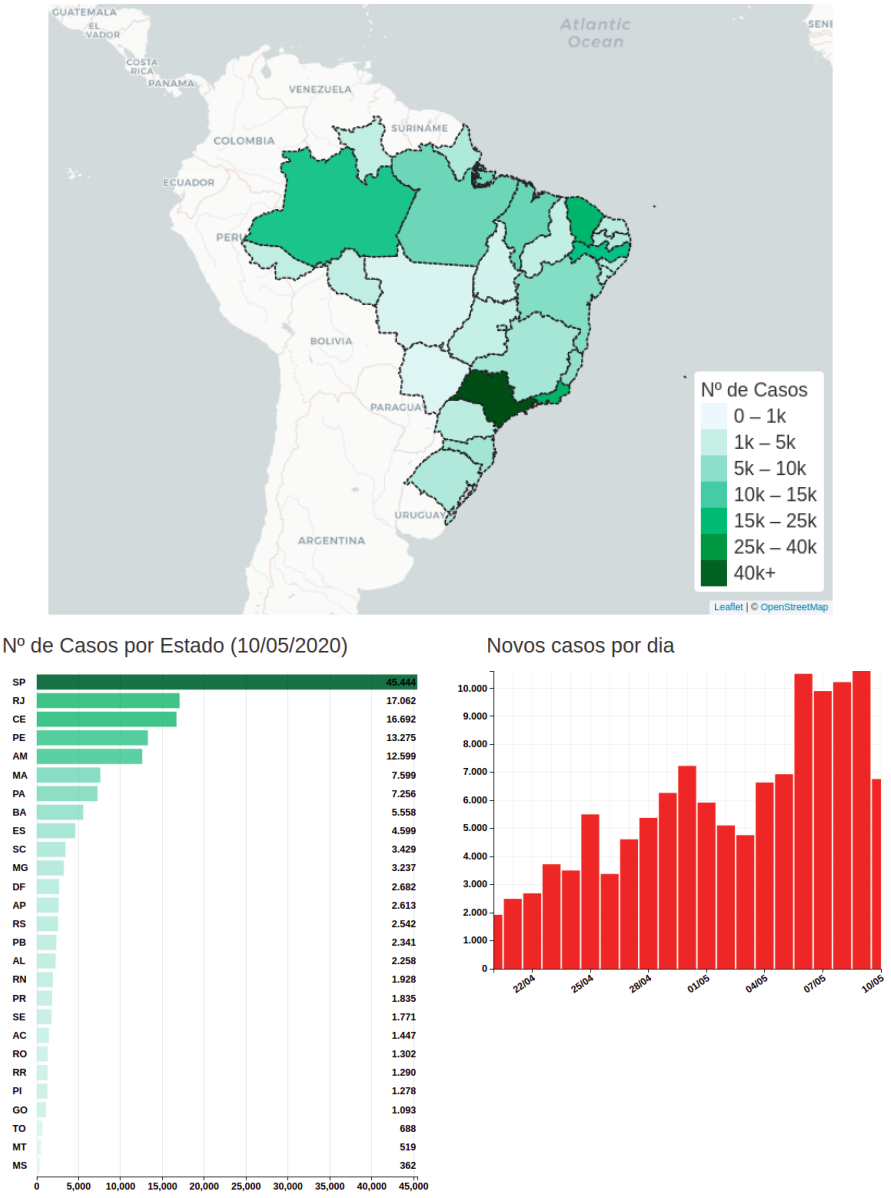

Figura 2.6. Ilustração das visualizações simplificadas da COVID-19 construída a partir dos Códigos 2.6, 2.7 e 2.8 , para demonstração das bibliotecas utilizadas e suas contribuições.

[28]Disponíveis aqui: `https://drive.google.com/drive/folders/11a--lk-HBLvhf5dtkczDNZBzfxbXWCn5?usp=sharing`

2.4. Conclusão

Este trabalho apresentou técnicas eficientes para aquisição, tratamento e visualizações interativas de dados, com ênfase nos dados disponibilizados pelo portal da COVID-19 do Ministério da Saúde. Assim, fornecemos subsídios para a utilização e construção de ferramentas práticas e eficazes que podem auxiliar a sociedade e seus governantes em problemáticas como as geradas pela pandemia da COVID-19.

Epidemias são diferentes de doenças isoladas, sua abrangência envolve diversos fatores e diferentes atores, uma vez que não se relacionada apenas à saúde das pessoas, mas também à política, economia, cultura, educação e outros aspectos de vida. Atualmente, métodos e técnicas como as apresentadas neste trabalho tornam-se cada vez mais importantes, tendo em vista que a quantidade de dados disponíveis é gigantesca. O conhecimento e domínio de tais técnicas podem auxiliar diferentes profissionais no desenvolvimento de soluções computacionais tempestivas e satisfatórias aos mais diversos problemas enfrentados no cotidiano da vida em sociedade.

Referências

[Cairo 2015] Cairo, A. (2015). *The Truthful Art: Data, Charts, and Maps for Communication*. New Riders.

[Datasus 2020] Datasus (2020). Corona virus brasil. `https://covid.saude.gov.br`. Acessado em 2020-05-09.

[Doyle 2020] Doyle, S. (2020). Migrant workers falling through cracks in health care coverage. *CMAJ*, 192(28):E819–E820.

[Government 2020] Government, N. Y. S. (2020). New york state covid-19 data. `https://data.ny.gov`. Acessado em 2020-05-09.

[Grus 2019] Grus, J. (2019). *Data Science do zero: Primeiras regras com o Python*. Alta Books.

[Knaflic 2015] Knaflic, C. N. (2015). *Storytelling with Data: A Data Visualization Guide for Business Professionals*. Wiley.

[McKinney 2012] McKinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. "O'Reilly Media, Inc.".

[Meeks 2015] Meeks, E. (2015). *D3.js in Action*. Manning Publications.

[Mitchell 2019] Mitchell, R. (2019). *Web Scraping com Python – 2ª edição: Coletando mais dados da web moderna*. NOVATEC.

[Reitz 2020] Reitz, K. (2020). requests: Python http for humans. *URL: http://python-requests. org*.