

**UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO  
BACHARELADO EM CIÊNCIAS DE DADOS**

**ANDRÉ THIAGO GOMES TABOADA RA 2003616**

**ANDREIA MARIA DA SILVA RAMOS RA 2015234**

**BIANCA DE SOUZA VIEIRA RA 2011157**

**CARLOS ADALBERTO SILVA MOREIRA JUNIOR RA 2015232**

**GABRIEL MERÊNCIO DOS SANTOS RA2000440**

**LAIZ HIROMI KODAIRA RA 2002395**

**LORENZO NATHANIEL NO RA 2001230**

**LUCAS RODRIGUES ZANFORLIN RA 2015835**

**Aplicando Modelos de Machine Learning e Deep Learning  
para disponibilização e visualização dos dados de  
sistemas produtores da SABESP via dashboard e API.**

< <https://youtu.be/uuZZEzIPUfk> >

SÃO PAULO, SP

Novembro, 2023

**UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO  
BACHARELADO EM CIÊNCIAS DE DADOS**

**ANDRÉ THIAGO GOMES TABOADA RA 2003616**

**ANDREIA MARIA DA SILVA RAMOS RA 2015234**

**BIANCA DE SOUZA VIEIRA RA 2011157**

**CARLOS ADALBERTO SILVA MOREIRA JUNIOR RA 2015232**

**GABRIEL MERÊNCIO DOS SANTOS RA2000440**

**LAIZ HIROMI KODAIRA RA 2002395**

**LORENZO NATHANIEL NO RA 2001230**

**LUCAS RODRIGUES ZANFORLIN RA 2015835**

**Aplicando Modelos de Machine Learning e Deep Learning  
para disponibilização e visualização dos dados de  
sistemas produtores da SABESP via dashboard e API.**

Projeto de pesquisa submetido como requisito para aprovação na disciplina Trabalho de Conclusão de Curso do curso de Bacharelado em Ciências de Dados da Universidade Virtual do Estado de São Paulo.

São Paulo, SP

Novembro, 2023.

TABOADA, André Thiago Gomes; RAMOS, Andreia Maria da Silva; VIEIRA, Bianca de Souza; JUNIOR, Carlos Adalberto Silva Moreira; SANTOS, Gabriel Merêncio dos; KODAIRA, Laiz Hiromi; NO, Lorenzo Nathaniel; ZANFORLIN, Lucas Rodrigues. **Disponibilização e visualização de dados de sistemas produtores da SABESP via dashboard e API.** 60f. Trabalho de Conclusão de Curso (Bacharelado em Ciência de Dados). Universidade Virtual do Estado de São Paulo, São Paulo, 2023.

## Resumo

Este trabalho propõe a disponibilização e visualização dos dados de sistemas produtores da SABESP via *dashboard* e API. O objetivo geral é tornar o acesso aos dados mais democrático, comprehensível e acessível por meio de uma API e *dashboard*. A SABESP disponibiliza dados sobre os sistemas produtores que abastecem o estado de São Paulo em um site próprio. No entanto, o acesso a esses dados é dificultado por alguns problemas, como a seleção lenta de intervalos grandes de tempo e a falta de formatos padronizados e convenientes, além da ausência de gráficos e uma apresentação acessível ao público geral. Para resolver esses problemas, o trabalho coleta os dados da SABESP e os disponibiliza em formatos padronizados e convenientes, além de disponibilizar um *dashboard* com os dados em gráficos interativos para o público geral. O trabalho contribui para a transparência e o controle social na gestão de recursos hídricos. A disponibilização dos dados de forma mais democrática, comprehensível e acessível permite que o público geral monitore a situação em sua região e que desenvolvedores, profissionais de dados, acadêmicos e outros públicos especializados possam contribuir com novas ferramentas, soluções, modelos e *insights*. Após a construção da aplicação, usando metodologias como *design thinking* e CRISP-DM, distribuímos um questionário ao público geral e obtivemos avaliações positivas.

Palavras-chave: disponibilização e visualização dos dados, análise de dados, dados públicos SABESP

## **Lista de ilustrações**

<b>Figura 1</b> – Erro de <i>Gateway Timeout</i> na página da SABESP.....	5
<b>Figura 2</b> – Arquivo XML exportado na página da SABESP.....	6
<b>Figura 3</b> – Tentativa de importar os dados da SABESP com a biblioteca <i>Pandas</i> .....	6
<b>Figura 4</b> – Tabulação original dos dados disponibilizados pela SABESP.....	7
<b>Figura 5</b> – Gráfico mostrando a vazão observada e prevista de um ponto de controle no SPHM-PJC.....	9
<b>Figura 6</b> – Página inicial do protótipo do <i>dashboard</i> .....	13
<b>Figura 7</b> – Página do Sistema Cantareira no protótipo do <i>dashboard</i> .....	13
<b>Figura 8</b> – Tabela com os dados selecionados e opção de <i>download</i> no protótipo do <i>dashboard</i> .....	14
<b>Figura 9</b> – Requisição do portal da SABESP aos dados dos sistemas.....	15
<b>Figura 10</b> – Visualização do <i>boxplot</i> .....	18
<b>Figura 11</b> – Colunas da Represa Cachoeira da França.....	20
<b>Figura 12</b> – Histograma dos valores do teste.....	22
<b>Figura 13</b> – <i>Analytical Base Table</i> (ABT).....	23
<b>Figura 14</b> – Modelo de Média (MSE).....	24
<b>Figura 15</b> – <i>Decision Tree Regressor</i> .....	25
<b>Figura 16</b> – <i>Random Forest Regressor</i> .....	26
<b>Figura 17</b> – <i>Random Forest Regressor</i> com ganho de performance.....	26
<b>Figura 18</b> – Valores do RMSE e do $R^2$ .....	27
<b>Figura 19</b> – Modelo <i>XGBoost</i> .....	28
<b>Figura 20</b> – Etapas do CRISP-DM.....	29
<b>Figura 21</b> – <i>Dashboard</i> final após melhorias.....	32
<b>Figura 22</b> – Página “Sobre os dados” na aplicação final.....	33
<b>Figura 23</b> – Página “API” na aplicação final.....	33
<b>Figura 24</b> – <i>Endpoints</i> da API na documentação.....	34
<b>Figura 25</b> – Documentação de um <i>endpoint</i> da API.....	35
<b>Figura 26</b> – Resultado de um teste de um <i>endpoint</i> da API.....	35
<b>Figura 27</b> – Descrição dos dados fornecidos pela API na documentação.....	36

<b>Figura 28</b> – Resumo das respostas às perguntas 1 e 2 do questionário de avaliação.....	38
<b>Figura 29</b> – Resumo das respostas às perguntas 3 e 4 do questionário de avaliação.....	39
<b>Figura 30</b> – Resumo das respostas às perguntas 4 e 5 do questionário de avaliação.....	40
<b>Figura 31</b> – Resumo das respostas às perguntas 6 e 7 do questionário de avaliação.....	41
<b>Figura 32</b> – Resumo das respostas às perguntas 8 e 9 do questionário de avaliação.....	42
<b>Figura 33</b> – Resumo das respostas à pergunta 10 do questionário de avaliação....	42

## **Lista de tabelas**

<b>Tabela 1</b> – Sistema Produtores e seus respectivos reservatórios.....	19
<b>Tabela 2</b> – Descrição das colunas.....	20

## **Sumário**

<b>1 Introdução .....</b>	<b>1</b>
<b>2 Objetivos .....</b>	<b>3</b>
2.1 Objetivos gerais .....	3
2.2 Objetivos específicos .....	3
<b>3 Justificativa.....</b>	<b>4</b>
<b>4 Referencial teórico .....</b>	<b>7</b>
4.1 Catalogação de referências .....	7
4.2 Trabalhos relacionados.....	8
4.3 Diferencial competitivo .....	10
<b>5 Metodologia .....</b>	<b>12</b>
5.1 Coleta de dados.....	14
5.2 Tratamento dos dados .....	15
5.3 Análise exploratória dos dados.....	18
5.4 Construção do dashboard.....	29
5.5 Construção da API.....	30
<b>6 Resultados e discussões .....</b>	<b>32</b>
<b>7 Conclusões .....</b>	<b>43</b>
<b>8 Referências .....</b>	<b>45</b>
<b>9 Apêndices .....</b>	<b>47</b>

## 1 Introdução

O acesso à informação é uma questão relevante e de suma importância quando consideramos nossos direitos como cidadãos. Podemos encontrar na Lei de Acesso à Informação, lei nº 12.527/2011, as garantias do nosso direito constitucional de solicitar e obter informações de órgãos e entidades públicas. O livre acesso aos dados e informações permitem que cidadãos possam cobrar e participar de forma mais ativa e, consequentemente, melhorar a gestão das instituições públicas. Podemos chamar isso de controle social, também assegurado pela Constituição Federal de 1988 (BEZERRA, 2021).

Desta forma, a partir da experiência do grupo em um projeto integrador anterior (VIEIRA et. al, 2023), verificou-se uma deficiência na disponibilização e acesso aos dados da Companhia de Saneamento Básico do Estado de São Paulo (SABESP). A SABESP é uma sociedade anônima de economia mista, sendo o estado de São Paulo o maior acionista, e por isso disponibiliza publicamente seus dados, serviços e prestação de contas. Contraditoriamente, apesar de conseguirmos acessar os dados da SABESP em seu próprio site, verificamos uma certa dificuldade na extração, manipulação e visualização dos dados, resultando em uma barreira para grande parte da população.

Somado a isso, temos que a mera disponibilização dos dados não garante à população acesso à informação. Como mencionado por Bezerra (2021):

A participação ativa do cidadão no controle social pressupõe a transparência das ações governamentais. O governo deve propiciar ao cidadão a possibilidade de entender os instrumentos de gestão, para que ele possa influenciar no processo de tomada de decisões. O acesso do cidadão à informação simples e compreensível é o ponto de partida para uma maior transparência. (BEZERRA, 2021)

O dado por si só não significa informação; assim como descrito por Jamil (2005), considera-se o dado como um fato, código ou sinal que pode ser obtido por observação ou medição, mas, isoladamente, sua compreensão é complexa. Para que haja informação, é necessário contextualização e dados suficientes para uma análise.

Em síntese, podemos considerar que a disponibilização dos dados, meramente dispostos em uma tabela, não nos gera informação. É necessário todo um trabalho de extração, tratamento e carregamento dos dados para posteriormente ser possível realizar uma análise e contextualização, gerando a informação, para em seguida ser disponibilizado de maneira que seja compreensível e mais acessível à população.

O foco do trabalho também foram os dados de represas e mananciais gerenciados, monitorados e preservados pela SABESP. São cinco sistemas produtores de água sob sua supervisão: Alto Cotia, Alto Tietê, Cantareira, Guarapiranga, Rio Claro, Rio Grande e São Lourenço. A SABESP é responsável por tratar e distribuir água potável para mais de 14 milhões de moradores da capital e grande São Paulo (SABESP, s.d). Essas 14 milhões de pessoas são diretamente afetadas, tanto positivamente quanto negativamente, pelas decisões tomadas sobre a gestão desses recursos. Por isso, é de suma importância que haja uma boa disponibilização dos dados e informações para que mais pessoas possam colaborar com *insights* e sugestões de melhorias nas mais diversas áreas de atuação da SABESP.

Desta forma, o trabalho a seguir realizou uma análise técnica com relação a maneira com que o acesso dos dados da SABESP se faz disponível à população. Tendo como base esta análise, desenvolvemos uma solução visando facilitar a democratização dos dados, tornando-o mais acessível a pesquisadores e estudiosos. Por fim, com o objetivo de atingir o público leigo, elaboramos um *dashboard* interativo, no qual é possível visualizar os dados referentes aos sistemas produtores da Sabesp.

## 2 Objetivos

### 2.1 Objetivos gerais

O objetivo geral do trabalho foi tornar o acesso aos dados de sistemas produtores da SABESP mais democrático, compreensível e acessível por meio de uma API e *dashboard*. O resultado final desejado foi um produto de dados de simples acesso que permita a desenvolvedores, profissionais de dados e o público geral monitorarem os dados e construírem trabalhos com base neles.

### 2.2 Objetivos específicos

Os objetivos específicos deste trabalho foram:

- Verificar e identificar a forma com que a SABESP atualmente disponibiliza os dados e gera informação;
- Disponibilizar os dados atualizados dos sistemas produtores da SABESP em formatos padronizados e convenientes em plataformas como GitHub e Kaggle;
- Construir uma API de acesso aos dados para desenvolvedores;
- Construir um *dashboard* com os dados de sistemas produtores em gráficos interativos para o público geral.

### 3 Justificativa

A Companhia de Saneamento Básico do Estado de São Paulo (SABESP) disponibiliza dados sobre os sistemas produtores que abastecem o estado de São Paulo em um site próprio<sup>1</sup>. A disponibilização desses dados é vital para a participação ativa da população na gestão de recursos hídricos, permitindo que o público geral monitore a situação em sua região e que desenvolvedores, profissionais de dados, acadêmicos e outros públicos especializados possam contribuir com novas ferramentas, soluções, modelos e *insights*.

O monitoramento desses dados é particularmente relevante no contexto de crises hídricas, como a crise de abastecimento de 2014 em São Paulo. Além dos problemas de saneamento básico e saúde pública, a crise gerou consequências econômicas devido à redução do ritmo de produção e queda de produtividade em indústrias, produzindo milhares de demissões (Redação RBA, 2014). Dado que o estado de São Paulo corresponde a um terço do Produto Interno Bruto do Brasil, a crise também se mostrou uma ameaça à recuperação econômica do país (Dezem, 2014).

Souza-Fernandes (2016) destaca a relevância da participação da sociedade na gestão de recursos hídricos:

A água é um problema de segurança nacional e como tal merece a adoção de estratégias direcionadas para cada um de seus aspectos particulares, todos eles de relevância para o desenvolvimento dos povos, aí compreendida a saúde pública. E todos devem se tornar participes nesta gestão. (SOUZA-FERNANDES, 2016, p. 95)

Considerando a participação da sociedade, há alguns problemas e pontos fracos no modo como a SABESP disponibiliza os dados de sistemas produtores, dificultando o acesso e uso dos dados. Do ponto de vista do público geral, os dados estão disponíveis apenas em formato tabular, o que não é conveniente para visualização, principalmente para o acompanhamento dos indicadores ao longo do

---

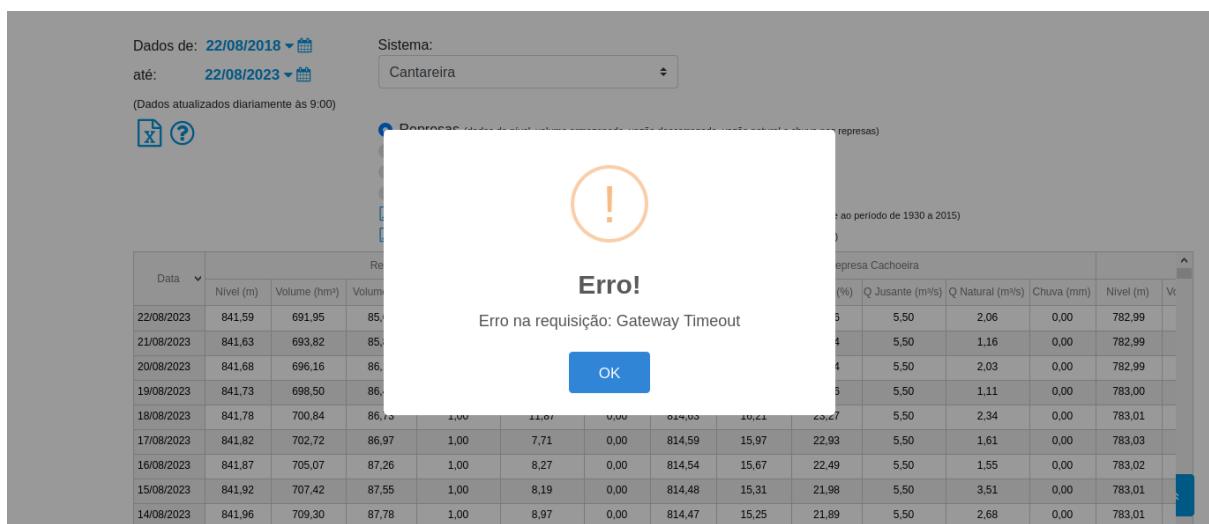
<sup>1</sup> <https://mananciais.sabesp.com.br/HistoricoSistemas?SistemaId=0>

tempo. Isso acaba sendo um ruído na questão da transparência e controle social, uma vez que a disponibilização dos dados não se dá de maneira comprehensível e facilitada.

Do ponto de vista de desenvolvedores e profissionais de dados, identificamos três problemas:

1. A seleção de intervalos grandes de tempo é lenta e pode falhar por completo, resultando em um *Gateway Timeout*, como no exemplo da figura 1;
2. Embora o arquivo com os dados exportados tenha a extensão .xls e possa ser aberto normalmente no Excel e outros editores de planilhas, ele é na verdade um arquivo XML (*Extensible Markup Language*) com uma estrutura própria, como exibido na figura 2. Isso dificulta a abertura e manipulação do arquivo em linguagens de programação como o *Python*. A figura 3 mostra uma tentativa inexitosa de abrir o arquivo exportado pelo portal da SABESP com a biblioteca *Pandas* para ilustrar essa dificuldade; o erro persiste independentemente da *engine* especificada, pois o formato não é reconhecido.
3. O formato da tabela com os dados por represa também é inconveniente para bibliotecas como o Pandas. Conforme mostra a figura 4, cada represa aparece como *header* e os dados de cada represa em um determinado dia aparecem em uma mesma linha. Normalmente é mais conveniente ter a represa como uma coluna, com dados de apenas uma represa por linha.

**Figura 1 – Erro de *Gateway Timeout* na página da SABESP**



Fonte: Portal dos Mananciais Sabesp<sup>2</sup>.

<sup>2</sup> <https://mananciais.sabesp.com.br/HistoricoSistemas?SistemaId=0>

**Figura 2 – Arquivo XML exportado na página da SABESP**

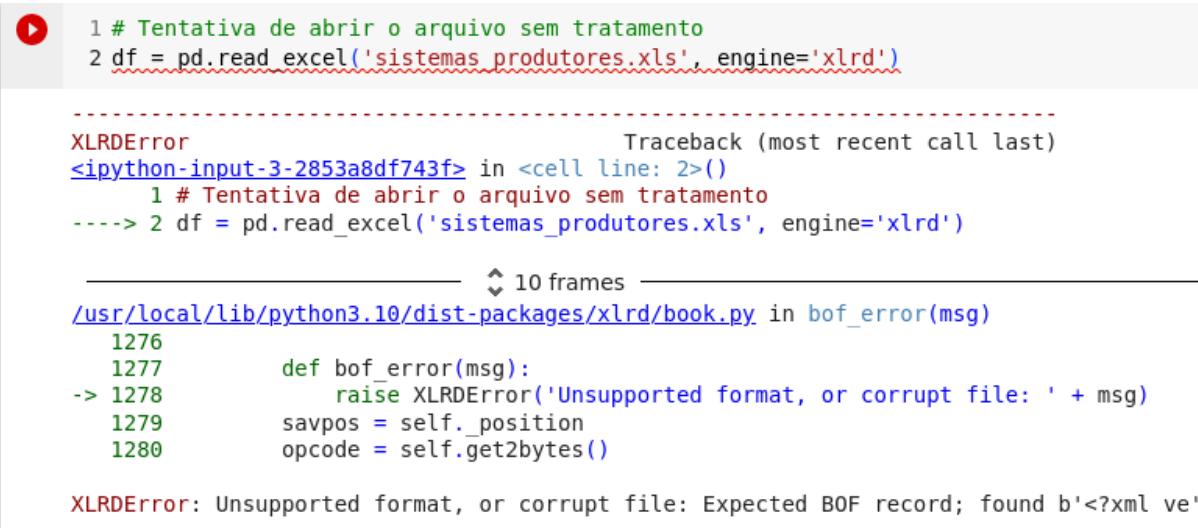


```
sistemas_produtores.xls

1 <?xml version="1.0"?>
2 <?mso-application progid="Excel.Sheet"?>
3 <Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
4 xmlns:office="urn:schemas-microsoft-com:office:office"
5 xmlns:x="urn:schemas-microsoft-com:office:excel"
6 xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
7 xmlns:html="http://www.w3.org/TR/REC-html40">
8 <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
9 <Version>12.00</Version>
10 </DocumentProperties>
11 <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
12 <WindowHeight>8130</WindowHeight>
13 <WindowWidth>15135</WindowWidth>
14 <WindowTopX>120</WindowTopX>
15 <WindowTopY>45</WindowTopY>
16 <ProtectStructure>False</ProtectStructure>
17 <ProtectWindows>False</ProtectWindows>
18 </ExcelWorkbook>
19 <Styles>
20   <Style ss:ID="xls-style-1" ss:Name="xls-style-1">
21     <Alignment ss:Vertical="Bottom" ss:Horizontal="Center"/>
22     <Borders>
23       <Border ss:Position="Bottom" ss:LineStyle="Continuous" ss:Weight="1" ss:Color="#CACACA"/>
24       <Border ss:Position="Left" ss:LineStyle="Continuous" ss:Weight="1" ss:Color="#CACACA"/>
25       <Border ss:Position="Right" ss:LineStyle="Continuous" ss:Weight="1" ss:Color="#CACACA"/>
26       <Border ss:Position="Top" ss:LineStyle="Continuous" ss:Weight="1" ss:Color="#CACACA"/>
27     </Borders>
28     <Font ss:Color="#767676" />
29     <Interior ss:Color="#F4F4F4" ss:Pattern="Solid"/>
30   </Style>
```

Fonte: Elaborada pelos autores (2023).

**Figura 3 – Tentativa de importar os dados da SABESP com a biblioteca Pandas**



```
1 # Tentativa de abrir o arquivo sem tratamento
2 df = pd.read_excel('sistemas_produtores.xls', engine='xlrd')

-----
```

XLRDError Traceback (most recent call last)  
<ipython-input-3-2853a8df743f> in <cell line: 2>()  
 1 # Tentativa de abrir o arquivo sem tratamento  
----> 2 df = pd.read\_excel('sistemas\_produtores.xls', engine='xlrd')

-----

10 frames -----  

```
/usr/local/lib/python3.10/dist-packages/xlrd/book.py in bof_error(msg)
1276
1277     def bof_error(msg):
-> 1278         raise XLRDError('Unsupported format, or corrupt file: ' + msg)
1279     savpos = self._position
1280     opcode = self.get2bytes()
```

XLRDError: Unsupported format, or corrupt file: Expected BOF record; found b'<?xml ve'

Fonte: Elaborada pelos autores (2023).

**Figura 4 – Tabulação original dos dados disponibilizados pela SABESP**

Data	Represa Jaguari/Jacareí							Represa Cachoeira						
	Nível (m)	Volume (hm³)	Volume (%)	Q Jusante (m³/s)	Q Natural (m³/s)	Chuva (mm)	Nível (m)	Volume (hm³)	Volume (%)	Q Jusante (m³/s)	Q Natural (m³/s)	Chuva (mm)		
21/08/2023	841,63	693,82	85,86	1,00	5,05	0,00	814,65	16,33	23,44	5,50	1,16	0,00		
20/08/2023	841,68	696,16	86,15	1,00	4,81	0,00	814,65	16,33	23,44	5,50	2,03	0,00		
19/08/2023	841,73	698,50	86,44	1,00	5,68	0,00	814,64	16,27	23,36	5,50	1,11	0,00		
18/08/2023	841,78	700,84	86,73	1,00	11,87	0,00	814,63	16,21	23,27	5,50	2,34	0,00		
17/08/2023	841,82	702,72	86,97	1,00	7,71	0,00	814,59	15,97	22,93	5,50	1,61	0,00		
16/08/2023	841,87	705,07	87,26	1,00	8,27	0,00	814,54	15,67	22,49	5,50	1,55	0,00		
15/08/2023	841,92	707,42	87,55	1,00	8,19	0,00	814,48	15,31	21,98	5,50	3,51	0,00		
14/08/2023	841,96	709,30	87,78	1,00	8,97	0,00	814,47	15,25	21,89	5,50	2,68	0,00		
13/08/2023	842,00	711,19	88,01	1,00	6,18	0,20	814,46	15,19	21,81	5,50	1,33	0,80		
12/08/2023	842,05	713,55	88,31	1,00	7,74	0,00	814,43	15,01	21,55	5,50	1,02	4,80		
11/08/2023	842,10	715,92	88,60	1,00	4,50	0,00	814,38	14,71	21,13	5,50	0,80	0,00		
10/08/2023	842,15	718,28	88,89	1,00	5,27	0,00	814,37	14,65	21,04	5,50	1,23	0,00		
09/08/2023	842,20	720,65	89,18	1,00	5,03	0,00	814,35	14,54	20,87	5,50	1,02	0,00		

Fonte: Portal dos Mananciais Sabesp<sup>3</sup>.

## 4 Referencial teórico

### 4.1 Catalogação de referências

Buscamos três referências em bases de dados para nortear o trabalho, sobretudo na parte técnica, nas bases de dados do Google Acadêmico. Tomamos como referencial teórico os três livros aqui citados com foco em tratamento e análise de dados utilizando a linguagem *Python*.

Wes McKinney, escritor do livro “*Python para análise de dados*” é uma figura proeminente na comunidade de análise de dados com *Python*, colaborandoativamente e de forma voluntária em melhorias e desenvolvimento de softwares open sources tendo como foco a computação analítica (MCKINNEY, 2023). Ele é o criador da biblioteca *Pandas*, uma das mais conhecidas e utilizadas bibliotecas para análise e tratamento de dados. Além de utilizar a biblioteca pandas em seu livro, ele também faz uso de outras bibliotecas *Python* para a melhor manipulação e tratamento dos dados, tais como: *Numpy*, *Matplotlib*, *Scikit-learn*, entre outras. De maneira prática e didática, o livro aborda desde as etapas de carga e armazenagem de dados até a visualização e modelagem de dados, incluindo também um capítulo com exemplos práticos com dados reais.

---

<sup>3</sup> <https://mananciais.sabesp.com.br/HistoricoSistemas?SistemaId=0>

Daniel Y Chen é um engenheiro de dados e pesquisador no Laboratório de Decisões Análíticas e Sociais do Instituto de Biocomplexidade da *Virginia Tech* nos Estados Unidos da América, além de professor de Ciências de Dados na plataforma educacional *DataCamp*. Seu livro, “Análise de dados com *Python* e *Pandas*”, possui a mesma temática e abordagem da nossa primeira referência, mas possui um foco maior e abrangente quando se trata de tratamento e visualização de dados, tornando-se um incremento em nossas pesquisas de base teórica e prática.

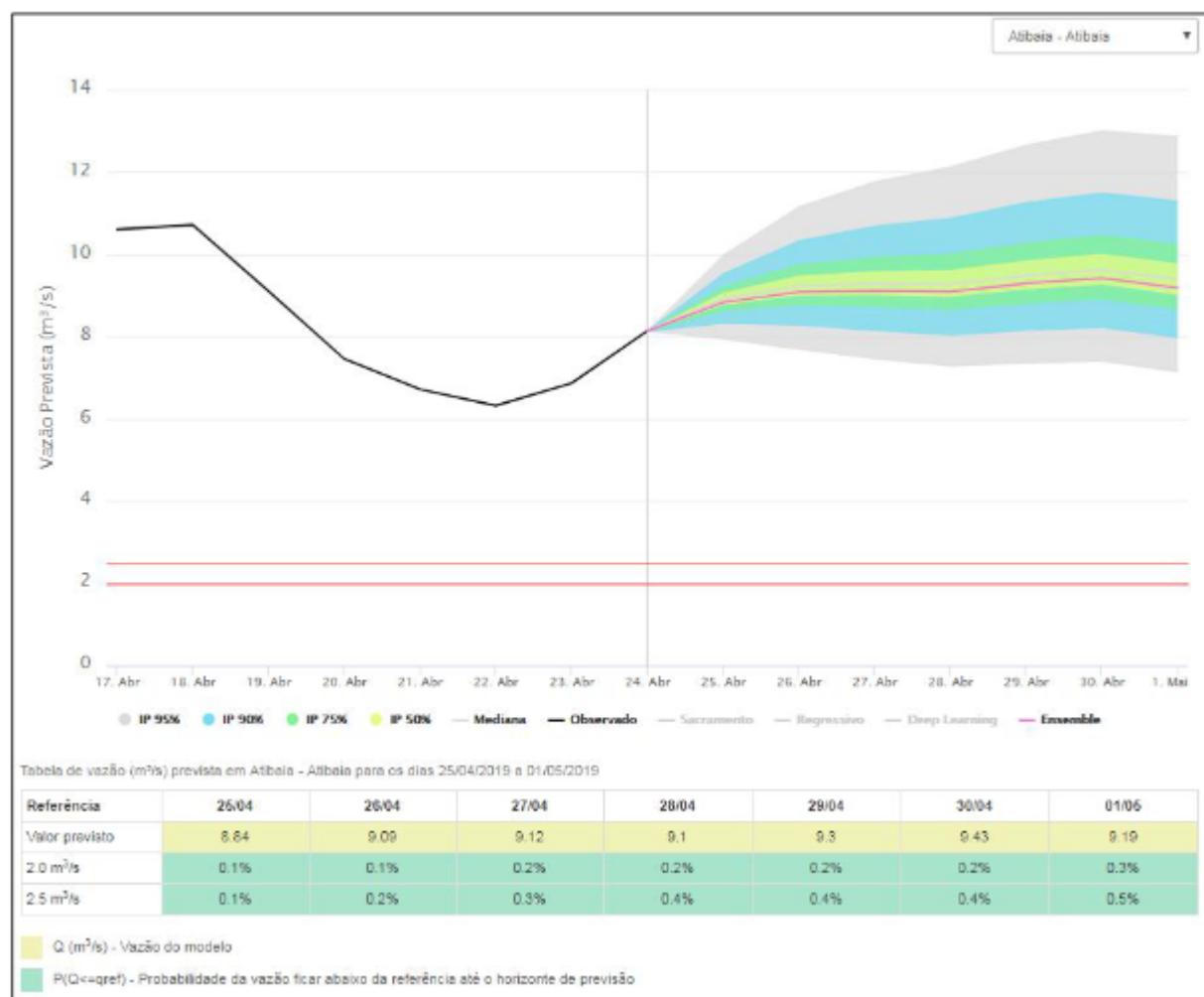
No Trabalho de Conclusão de Curso do curso de Engenharia de Produção da Universidade Federal Fluminense, Philipe Formigoni desenvolve uma análise de dados fazendo uso de Python e da metodologia CRISP-DM (*Cross Industry Standard Process for Data Mining*). Por se tratar de um Trabalho de Conclusão de Curso recente e com uma proposta próxima de nosso trabalho, achamos relevante considerar algumas das abordagens e aprendizados de Formigoni para enriquecer nossa análise de dados.

#### 4.2 Trabalhos relacionados

Além do referencial teórico usado como base, também pesquisamos trabalhos relacionados à disponibilização de dados da SABESP na literatura, em repositórios do GitHub e na internet em geral usando palavras-chave como “SABESP API”, “SABESP dashboard”, “SABESP visualização”, etc. Identificamos quatro trabalhos relevantes que estudamos para entender o que já foi feito, quais métodos foram empregados, quais são suas limitações e como o nosso trabalho pode se diferenciar.

Almeida et. al (2019) descrevem o Sistema de Previsão Hidrometeorológico das Bacias PCJ (SPHM-PJC), um sistema de visualização e previsão disponível para consulta dos técnicos da Agência das Bacias PCJ e da CT-MH dos Comitês PCJ. O sistema não é disponível para o público geral, além de ser restrito apenas às Bacias PCJ do Sistema Cantareira. A figura 5 mostra uma visualização do sistema.

**Figura 5** – Gráfico mostrando a vazão observada e prevista de um ponto de controle no SPHM-PJC



Fonte: Almeida et. al (2019).

O app Sabesp Mananciais RMSP<sup>4</sup> é um aplicativo móvel de monitoramento dos mananciais da Região Metropolitana de São Paulo disponibilizado pela SABESP ao público geral para dispositivos Android e iOS. Uma limitação é não haver uma versão web que funciona em qualquer dispositivo, o que diminui a acessibilidade. Também não é possível especificar um intervalo de tempo.

Presbiteris (2015) apresenta uma API que disponibiliza os dados dos reservatórios da SABESP. Contudo, além do fato de que a API não está mais disponível, ela disponibiliza apenas os dados de um dia por vez, não permitindo o

<sup>4</sup> <https://mananciais.sabesp.com.br/appsabesp>

acesso aos dados de um determinado intervalo. A API também parece disponibilizar apenas os dados do Sistema Cantareira.

Milz (s.d.) apresenta um pacote escrito na linguagem R que disponibiliza os dados de volume e pluviometria dos sistemas monitorados pela SABESP. Embora a base de dados completa esteja disponível para *download*, só é possível fazer requisições caso o desenvolvedor use a linguagem R. Também não há dados de reservatórios específicos.

Além desses quatro trabalhos, mencionamos também o trabalho desenvolvido por alguns dos integrantes deste grupo como parte da disciplina Projeto Integrador IV, cujo foco foi a modelagem preditiva do Sistema Cantareira (como descrito por Vieira *et. al* (2023)). Uma das dificuldades enfrentadas foi a etapa de extração e tratamento dos dados, que demandou mais tempo do que o previsto e foi uma das motivações para este trabalho. O grupo também não conseguiu implementar a atualização dos dados e das previsões diariamente de acordo com os novos dados fornecidos pela SABESP devido à limitação de tempo.

#### 4.3 Diferencial competitivo

Analizando os trabalhos relacionados, notamos algumas lacunas que nosso trabalho poderia preencher. Por exemplo, alguns dos sistemas não estão disponíveis ao público, como o descrito por Almeida *et. al* (2019), ou não são mais funcionais, como o descrito por Presbiteris (2015). Além disso, de modo geral, os projetos são focados na disponibilização de uma API ou na visualização dos dados, mas não em ambos.

Como diferencial competitivo, decidimos disponibilizar dados completos e atualizados via API, incluindo dados dos sistemas e dos reservatórios. Esses dados são disponibilizados via requisições HTTP à API, permitindo que desenvolvedores accessem os dados através de qualquer linguagem de programação. Em relação aos usuários, decidimos disponibilizar o *dashboard* via ambiente *web*, tornando-o

acessível através de qualquer dispositivo (ao contrário, por exemplo, do aplicativo da SABESP, que não é compatível com todos os dispositivos).

O projeto desenvolvido como parte do Projeto Integrador IV (VIEIRA *et. al.*, 2023) mostra uma das potenciais aplicações deste trabalho, pois a existência de um repositório ou API com os dados da SABESP atualizados agiliza o desenvolvimento de projetos de análise de dados e aprendizado de máquina, permitindo que pesquisadores e profissionais da área foquem na modelagem em vez da extração e tratamento dos dados.

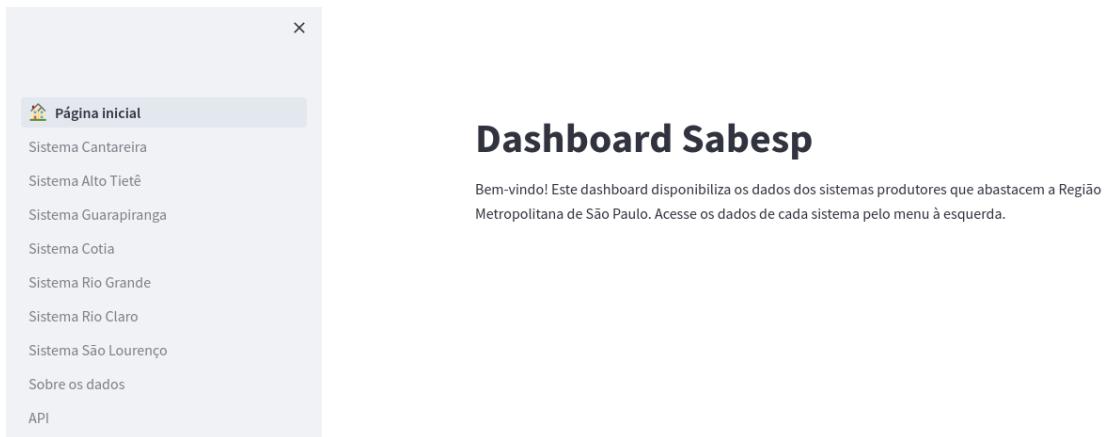
## 5 Metodologia

Nosso trabalho passou pelas etapas de revisão bibliográfica (descrita anteriormente), coleta dos dados de sistemas produtores da SABESP, construção do *dashboard* e construção da API. Como metodologia para o desenvolvimento do produto final, optamos pelo *design thinking*.

Segundo Brown e Wyatt (2010), o *design thinking* prevê processos interativos formados por três etapas, não necessariamente sequenciais: inspiração, ideação e implementação. A inspiração começa com os requisitos básicos do projeto (por exemplo, segmento de mercado e tecnologias disponíveis) e parte para o entendimento das necessidades do público-alvo. A ideação consiste na geração de ideias para o desenvolvimento do produto ou à resolução do problema. Na implementação, elabora-se um protótipo que é testado e refinado até a construção do produto final. A vantagem do *design thinking* sobre os processos tradicionais é que as etapas não são rigidamente definidas e abrem espaço para a criatividade e *feedback* contínuo.

O grupo seguiu as etapas de inspiração e ideação para definir o tema, os objetivos e os problemas que desejamos resolver através de discussões internas e conversas com a orientadora. Em seguida, desenvolvemos um protótipo do *dashboard* com uma ideia inicial do *design*. A figura 6 mostra a página inicial do protótipo, exibindo uma mensagem introdutória, com um menu lateral que permite ao usuário selecionar um sistema, acessar mais informações sobre os dados e ser direcionado à documentação da API.

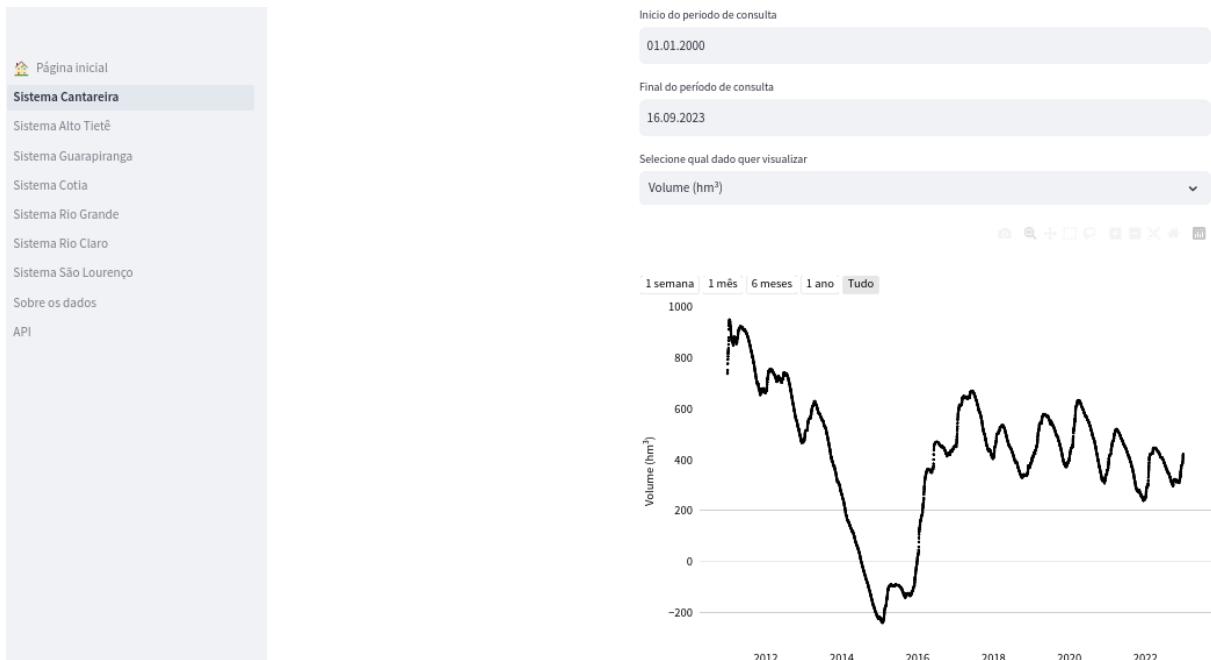
**Figura 6** – Página inicial do protótipo do *dashboard*



Fonte: Elaborada pelos autores (2023).

A figura 7 mostra a página do Sistema Cantareira, que é semelhante às outras páginas de sistemas produtores; nela, o usuário pode selecionar um período de consulta e qual dado deseja visualizar, gerando um gráfico interativo. Logo abaixo, como mostra a figura 8, o usuário encontra os dados em formato tabular e pode fazer *download* dos mesmos em formato CSV(Comma Separated Values, em português, Valores Separados por Vírgula).

**Figura 7** – Página do Sistema Cantareira no protótipo do *dashboard*



Fonte: Elaborada pelos autores (2023).

**Figura 8** – Tabela com os dados selecionados e opção de download no protótipo do dashboard



	Data	Volume (hm³)	Volume (%)	Chuva (mm)	Vazão natural (m³/s)	Vazão a jusante (m)
0	2011-01-01 00:00:00	735.9763	74.9412	0.1	30.346	
1	2011-01-02 00:00:00	739.4875	75.2987	36.25	70.475	
2	2011-01-03 00:00:00	749.6823	76.3368	50.1	147.125	4
3	2011-01-04 00:00:00	774.7399	78.8883	43.95	321.239	2
4	2011-01-05 00:00:00	791.3338	80.578	8.2	223.21	
5	2011-01-06 00:00:00	802.947	81.7605	13.35	168.225	
6	2011-01-07 00:00:00	814.6222	82.9493	13.15	168.489	
7	2011-01-08 00:00:00	822.6097	83.7627	3.8	125.485	2
8	2011-01-09 00:00:00	829.3022	84.4441	8.9	110.511	
9	2011-01-10 00:00:00	834.6583	84.9895	9.4	95.058	

[Baixar os dados como .csv](#)

Fonte: Elaborada pelos autores (2023).

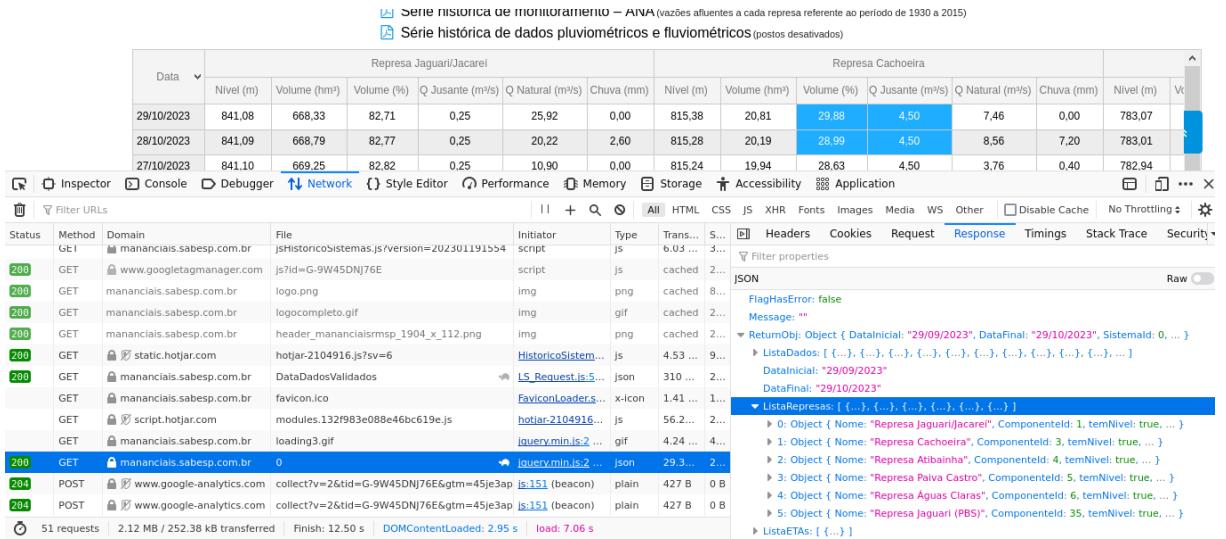
Os próximos passos do trabalho foram o refinamento sucessivo do protótipo e a implementação dos requisitos técnicos necessários para a validação final do trabalho.

## 5.1 Coleta de dados

O processo de coleta de dados foi conduzido por meio da prática de *web scraping*, um método amplamente adotado no âmbito da recuperação de informações online. De acordo com Mitchell (2019), o *web scraping* envolve o uso de programas automatizados para acessar páginas da web, requisitar dados e extrair informações. Essa técnica ganhou popularidade na primeira década dos anos 2000 com a popularização de navegadores baseados em um *web-kits opensource* e o surgimento de frameworks como o BeautifulSoup, os quais possibilitam a análise de arquivos HTML para a extração de dados de maneira mais eficiente.

Explorando o portal da SABESP com as ferramentas do navegador, notamos que a página faz uma requisição ao servidor para obter os dados, como exibido na figura 9. Assim, para simplificar a extração dos dados, criamos um programa que simula as requisições feitas pelo portal. Caso o modo como o portal obtenha os dados mude no futuro, podemos alterar o código ou usar um *framework* de automação como o Selenium.

**Figura 9 – Requisição do portal da SABESP aos dados dos sistemas.**



Fonte: Elaborada pelos autores (2023).

Compilamos todos os dados desde o ano 2000 e implementamos uma rotina de extração diária através do GitHub Actions<sup>5</sup>, uma plataforma que permite a execução automática de tarefas de acordo com as condições impostas. Configuramos um *script* para verificar novos dados aproximadamente às 09:00, 10:00, 15:00 e 20:00 no horário de Brasília como medida de contingência, já que o *script* pode falhar ou dados podem não estar disponíveis no horário indicado. Não há custos associados à extração, já que o plano gratuito do GitHub é suficiente para este projeto.

## 5.2 Tratamento dos dados

A etapa de tratamento de dados é fundamental para todo projeto de análise de dados, é possível verificar todas as etapas do tratamento dos dados no Apêndice A no final deste trabalho. Segundo Costa (2020), é uma das etapas mais demoradas de um projeto, que envolve atividades como limpeza, transformação e até a criação de novas variáveis com a finalidade de nos trazer mais informações de maneira eficiente. Costa (2020) também ressalta a importância desta etapa pelo fato de que, ao trabalharmos com diferentes fontes de dados, é comum nos

<sup>5</sup> <https://github.com/features/actions>

deparamos com erros, falta de padronização, valores nulos, redundâncias, etc. McKinney, também comenta que:

Durante a análise e a modelagem dos dados, um período significativo de tempo é gasto em sua preparação: carga, limpeza, transformação e reorganização. Sabe-se que essas tarefas em geral ocupam 80% ou mais do tempo de um analista. Às vezes, o modo como os dados são armazenados em arquivos ou em banco de dados não constituem um formato correto para uma tarefa em particular (MCKINNEY, 2017, p.358).

Como discutido anteriormente e ilustrado nas figuras 2 e 3, o arquivo exportado pelo site da SABESP está no formato XML e precisa passar por algum tratamento para que os dados possam ser lidos. Dessa forma, realizou-se uma transformação dos dados do formato XML para um *Dataframe* através da biblioteca de manipulação dos dados *Pandas*. De acordo com MCKINNEY(2017), um *DataFrame*:

[...] representa uma tabela de dados retangular e contém uma coleção ordenada de colunas, em que cada uma pode ter um tipo de valor diferente( numérico, string, booleano, etc.) O *DataFrame* tem índice tanto para linha quanto para coluna; pode ser imaginado como um dicionário de Series, todos compartilhando o mesmo índice. Internamente, os dados são armazenados como um ou mais blocos dimensionais em vez de serem armazenados como uma lista, um dicionário ou outra coleção de arrays unidimensionais(MCKINNEY, 2017, p. 248)

Outro ponto a ser importante a ser lembrado é as adequações necessárias para um melhor entendimento e estruturação dos dados de um *Dataframe* ( Costa, 2020). Em nosso caso, foi preciso adicionar uma coluna “Represa” ao *Dataframe* para especificar a origem dos dados apresentados.

Adicionalmente, como é impossível extrair todos os dados de uma vez por limitação do site da SABESP, foi preciso criar um *DataFrame* por ano e posteriormente foi realizada sua concatenação por meio da função *concat*. Para melhor visualização dos dados no arquivo CSV bruto, realizou-se a ordenação dos dados tendo como parâmetro a coluna ‘Data’.

Com todos os dados desde 2000 até a data da extração compilados em arquivos CSV, verificamos a consistência e integridade dos dados e realizamos tratamentos adicionais quando necessário. Através da função *info*, foi possível

visualizar as colunas existentes, a quantidade de colunas não nulas e os tipos dos dados no *Dataframe*. Após essa verificação foi preciso realizar a conversão dos dados na coluna ‘Data’ através da função *datetime*, uma vez que ela se apresentou no formato *object*.

Assim como descrito por Costa (2020) em toda análise de dados se faz necessário a verificação da presença de dados ausentes para que possamos garantir “qualidade, completude, veracidade e integridade dos fatos que aqueles dados representem”. Chen (2018) cita que dificilmente encontraremos um conjunto de dados sem dados ausentes ou nulos, sendo que elas podem se apresentar como *null*, *nan*, *na* e até como strings vazias ‘ ’. Desta forma, o tratamento dos valores nulos dependerá de uma análise específica dos dados, buscando identificar os motivos, problemas ou possíveis distorções (MCKINNEY, 2017, p. 360). Especificamente ao nosso projeto, após análise de cada caso, foi verificado a ausência de valores nulos, não sendo preciso realizar nenhum tipo de tratamento.

Dando sequência ao tratamento de nosso *Dataframe*, é essencial que não haja duplicidade de dados, tanto nas colunas quanto nas linhas, desta forma, foram utilizadas as funções *duplicate* e *drop\_duplicate*, a primeira função para verificar se há duplicidade e o segundo para remoção dos dados duplicados, caso existentes (MCKINNEY, 2017).

Por fim, foi verificado também a existência de *outliers*, dados fora do padrão ou discrepantes, em nossos conjuntos de dados. Acerca dos *outliers*, Costa (2020), diz que:

[...] outliers são instâncias ou valores de um atributo que são significativamente inconsistentes ou diferentes, ou seja, valores fora do padrão de um determinado conjunto de dados. Portanto, detectar desvios objetiva identificar mudanças em padrões que já foram percebidos. A identificação de outliers, ou detecção de desvios, pode ajudar a solucionar problemas da empresa. (COSTA, 2020, p. 109)

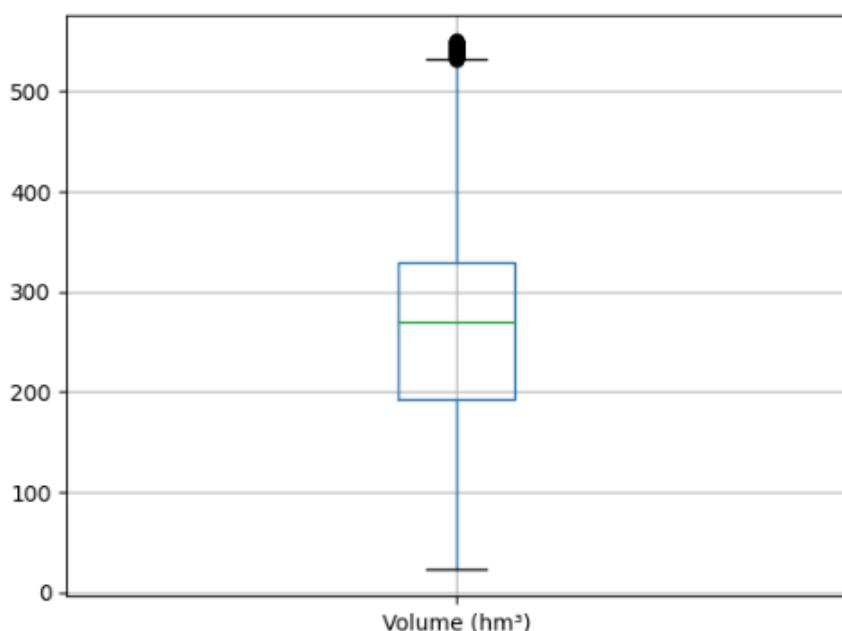
Para tal verificação, foi utilizado o gráfico de caixa, também conhecido como *boxplot*. Sobre o *boxplot* e os *outliers*, Galarnyk (2018), citado por Costa (2020, p. 111), diz que “esse tipo de gráfico *boxplot* organiza os dados em grupos por quartis,

bem como traça a variabilidade abaixo ou acima dos quartis inferiores e superiores, respectivamente. Além disso, coloca os *outliers* como pontos fora da caixa". Desta forma, a verificação da presença de *outliers* se faz necessária, principalmente para evitar que não haja distorções futuras no desenvolvimento da análise dos dados. Na figura 10, temos a representação visual do *boxplot*.

**Figura 10** - Visualização do *boxplot*.

```
[ ] df_alto_tiete.boxplot(column="Volume (hm³)")
```

<Axes: >



Fonte: Elaborada pelos autores (2023).

### 5.3 Análise exploratória dos dados

O objetivo inicial da modelagem era a criação de modelos que predissessem o nível de água nos sistemas de abastecimento da SABESP. Ao analisar o site dos sistemas produtores da SABESP, identificamos que temos um total de sete sistemas de abastecimento: Alto Tietê, Cantareira, Cotia, Guarapiranga, Rio Claro, Rio Grande e São Lourenço. Cada sistema é composto por uma certa quantidade de represas que fluem e se unificam no sistema de abastecimento compondo o seguinte sistema:

**Tabela 1** – Sistema Produtores e seus respectivos reservatórios

Sistemas Produtores	Reservatórios
Alto Tietê	Paratinga, Ponte Nova, Biritiba, Jundiaí, Taiaçupeba e Biritiba
Cantareira	Jaguari/Jacareí, Cachoeira, Atibainha, Paiva Castro, Águas Claras e Jaguari (JBS)
Cotia	Pedro Beichte, Represa da Graça e Isolina
Guarapiranga	Guarapiranga, Capivari e Billings
Rio Claro	Ribeirão do Campo
Rio Grande	Rio Grande, Ribeirão da Estiva
São Lourenço	Cachoeira da França

Fonte: Elaborada pelos autores (2023).

Ao realizar a análise inicial das tabelas, percebemos que algumas não continham todas as *features* para a elaboração de um modelo de Machine Learning. Dessa forma, para uma análise minuciosa, optamos por realizar a modelagem dos dados para o Sistema São Lourenço, que é composto por apenas um reservatório. Caso a escolha fosse os demais sistemas, teríamos que criar um modelo por represa dado que cada uma possui suas particularidades específicas.

**Figura 11** - Colunas da Represa Cachoeira da França.

Data	Represa Cachoeira do França					
	Nível (m)	Volume (hm <sup>3</sup> )	Volume (%)	Q Jusante (m <sup>3</sup> /s)	Q Natural (m <sup>3</sup> /s)	Chuva (mm)

Fonte: Portal das Mananciais Sabesp.

Destacamos ainda que foi incluído um padrão para nomeação das colunas sendo mnemônicos no início da cara coluna do Dataset e CamelCase. As colunas foram transformadas para:

"VrNivel", "VIVolumeHm", "PcVolume%", "VzQJusante", "VzQNatural", "VrChuva", "DdDia", "MmMes", "AaAno".

**Tabela 2 – Descrição das colunas**

Mnemonics	Significado
Vr	Colunas onde a informação do seu dado se refere a algum tipo de valor, seja int, float etc.
Vz	Colunas onde a informação remete a vazão
Pc	Percentual % da informação
VI	Volume hm <sup>3</sup> (Hectare cúbico)

Fonte: Elaborada pelos autores (2023).

Na coluna data foi realizado um split para quebra em 3 nas colunas, AaAno, MmMes e DdDia.

O objetivo inicial do desenvolvimento do modelo de previsão do nível do reservatório de São Lourenço foi o uso de uma regressão linear. Para isso, foi necessário realizar um tratamento dos dados para atender às premissas do modelo. Os próximos passos foram o enriquecimento da base de dados com a inclusão de

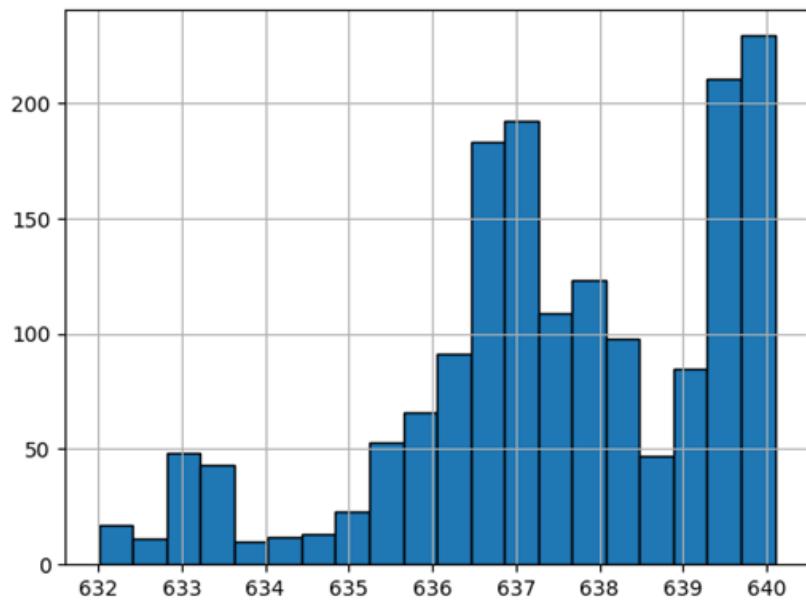
dados do INMET e do CIIAGRO. Esses dados fornecem informações sobre as condições climáticas e meteorológicas da região, que podem influenciar o nível do reservatório.

Para a análise exploratória dos dados, a ideia inicial era a criação de um modelo paramétrico de regressão linear. Esses modelos possuem determinadas premissas que precisam ser contempladas. Portanto, logo na análise exploratória dos dados, além dos *outliers*, nulos, missings e duplicados já observados no tratamento de dados, também verificamos a distribuição dos dados e sua normalidade.

Nesse caso, de acordo com Miot, H. A. (2017), uma solução apropriada é o teste de Shapiro-Wilk, que permite verificar se um conjunto de dados segue uma distribuição normal, o que é fundamental em estatística. A distribuição normal é frequentemente pressuposta por muitas técnicas estatísticas, e a validade dessa suposição é essencial para a interpretação precisa dos resultados. Além disso, o teste de Shapiro-Wilk ajuda a identificar valores atípicos e anomalias nos dados, auxilia na escolha do método estatístico correto e melhora a qualidade das previsões em várias aplicações. Em resumo, é uma ferramenta crucial para garantir a confiabilidade das análises estatísticas e a tomada de decisões informadas.

Dessa maneira, submetemos o código ao teste para compreender a distribuição dos dados e logo foi identificado que nenhuma das features possuía uma distribuição normal; para essa afirmação utilizados o plot de histogramas e o teste de Shapiro-Wilk que revelou valor P inferior a 0.05 para todas as variáveis.

**Figura 12** – Histograma dos valores do teste.



Fonte: Elaborada pelos autores (2023).

De acordo com os resultados, a base de dados apresenta um resultado de valor P inferior a 0,05 no teste de Shapiro-Wilk, o que indica que os dados não seguem uma distribuição normal. Como nossos dados retornaram valor P inferior a 0,05, precisamos fazer sua normalização, e para isto usamos o *Box-Cox Power Transform* e *MinMaxScaler*, porém tivemos muitos erros, já que todos dados precisavam ser convertidos, resultando em *overfit*. Optamos, portanto, pela escolha de modelos não-paramétricos. Vários modelos de regressão foram treinados, incluindo Média, *Decision Tree Regressor*, *Random Forest Regressor* e *XGBoost*.

Seguimos até a criação da ABT (Analytical Base Table), que é a base para iniciar a modelagem com os dados já tratados e processados. Ela é uma tabela que reúne dados de diversas fontes para serem utilizados em análises, uma etapa fundamental no processo de análise de dados, pois permite que os dados sejam organizados e estruturados de forma a facilitar a análise. A criação de uma ABT envolve as seguintes etapas:

1. Definição dos objetivos da análise: O primeiro passo é definir os objetivos da análise. Isso ajudará a determinar quais dados são necessários e como eles devem ser organizados.
2. Coleta de dados: Os dados podem ser coletados de diversas fontes, como sistemas de informação, arquivos, formulários e entrevistas.

3. Limpeza e preparação dos dados: Os dados devem ser limpos e preparados para a análise. Isso inclui a remoção de dados nulos, duplicados e incorretos.
4. Organização dos dados: Os dados devem ser organizados de forma a facilitar a análise. Isso pode ser feito através de tabelas, gráficos ou outros recursos visuais.

**Figura 13 – Analytical Base Table (ABT)**

```

def create_abt(tb1: pd.DataFrame, tb2: pd.DataFrame, cols_to_transoform: List[str]):
    df = pre_process_df1(df_raw = df1, date_col_name="i»/Data", cols_to_float =["Nível (m)", "Volume (hm³)", "Volume (%)", "Q Jusante (m³/s)"])
    #outliers
    df = df.drop(df[(df.VrUmidadeArMin < 10)].index)
    df = df.drop(df[(df.VrUmidadeArMax <= 93)].index)
    df = df.drop(df[(df.VrUmidadeArMed < 50)].index)
    df = df.drop(df[(df.VrPrecipitacao > 2)].index)
    df = df.drop(df[(df.VrTempMin <= 2)].index)
    df.drop(df[((df.VrTempMed < 2) | (df.VrTempMed >= 30))].index)
    df = df.drop(df[((df.VrTempMax < 0) | (df.VrTempMax >= 40))].index)
    df = df.drop(df[(df.VrNivel <= 629)].index)
    df = df.drop(df[(df.VlVolume < 0)].index)
    df = [variable: df: DataFrame] >= 35).index)
    df = [variable: df: DataFrame] < -2) | (df.VzQNatural >= 30)).index)
    df = df.drop(df[(df.VrChuva > 2)].index)
    #Regra utilizada: média do mesmo mês do ano anterior, e, caso não exista ano anterior, preenche com a média geral de todos os anos.
    #ordena df
    df = df.sort_values(by=['AaAno', 'MmMes'])

    for coluna in cols_to_transoform:
        if df[coluna].isnull().any():
            # média dos valores não nulos do ano anterior e mesmo mês
            media_mes_ano_anterior = df.groupby(['AaAno', 'MmMes'])[coluna].shift().groupby(df['AaAno']).transform('mean')

            # média geral de todos os anos para os valores não nulos
            media_geral = df[coluna].mean()

            # preenche os valores NaN na coluna específica com a média do ano e mês anterior,
            # e, caso não exista ano anterior, preencha com a média geral
            df[coluna] = df[coluna].fillna(media_mes_ano_anterior.fillna(media_geral))

    return df

```

Fonte: Elaborada pelos autores (2023).

Acerca dos modelos de *machine learning*, segue uma breve descrição, funcionamento e utilização de cada um utilizado após a separação dos dados em treino e teste:

- Média (RMSE - Root Mean Square Error)

Usamos como base para os modelos não-paramétricos o modelo da média. Tivemos o resultado do RMSE de +/- 2,37 em relação ao valor previsto. O modelo de média é um modelo simples de *machine learning* que pode ser usado para prever o valor médio de uma variável em um conjunto de dados. Ele é baseado na suposição de que os dados são distribuídos normalmente.

**Figura 14 – Modelo de Média (MSE)**

```
+ Código + Markdown | ▶ Executar Tudo Limpar Todas as Saídas >
+ ▶
    modelo_media_mse = statistics.mean((df['VrNivel']) - statistics.mean(df['VrNivel']))
    modelo_media_rmse = math.sqrt(modelo_media_mse)
    modelo_media_rmse
[725] ✓ 0.0s
...
... 2.372469305823453
+ ▶
    tree_model = DecisionTreeRegressor(random_state=42)
    tree_model.fit(x_train, y_train)
[630]
```

Fonte: Elaborada pelos autores (2023).

- *Decision Tree Regressor*

O *Decision Tree Regressor* é um modelo de árvore de decisão usado para problemas de regressão. O funcionamento do modelo é dividir os dados em subconjuntos com base em decisões em nós, com o objetivo de minimizar a variância da resposta (*target*) em cada subconjunto.

**Figura 15 – Decision Tree Regressor**

```

tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(x_train, y_train)

DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)

y_pred = tree_model.predict(x_test)

r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f'rmse:{rmse}, r2:{r2}')

rmse:1.0135277508178326, r2:0.8062412796948224

```

Fonte: Elaborada pelos autores (2023).

A avaliação do modelo foi feita usando as métricas RMSE e R<sup>2</sup>. O RMSE (*Root Mean Square Error*) é uma medida de erro quadrático médio, que indica a distância média entre os valores reais e os valores previstos pelo modelo. O R<sup>2</sup> (coeficiente de determinação) é uma medida de correlação entre os valores reais e os valores previstos pelo modelo, que varia de 0 a 1, sendo 1 o valor ideal. Os resultados do modelo *Decision Tree Regressor* foram os seguintes:

RMSE:	1,013
R <sup>2</sup> :	0,806

- *Random Forest Regressor*

O *Random Forest Regressor* é uma extensão do modelo de árvore de decisão, onde várias árvores são treinadas e a resposta final é a média ou moda das respostas individuais. O modelo busca reduzir a variância, fornecendo mais estabilidade e reduzindo o *overfitting*.

**Figura 16 – Random Forest Regressor**

```

rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(x_train, y_train)
y_pred = rf_model.predict(x_test)

r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f'rmse:{rmse}, r2:{r2}')

rmse:0.7329208001903514, r2:0.8986779278706971

```

Fonte: Elaborada pelos autores (2023).

Fazendo a imputação dos dados de 14 para as 6 principais variáveis observa-se um ganho de performance para as árvores aleatórias, conforme figura 17. Isso demonstra que, para o *machine learning*, trabalhar com menos variáveis com maior explicabilidade aprimora o resultado do algoritmo. O RMSE que mostra o erro quadrático médio diminuiu e o R<sup>2</sup> aumentou, demonstrando um melhor entendimento dos dados pelo modelo.

**Figura 17 – Random Forest Regressor com ganho de performance**

```

forest_model = RandomForestRegressor(random_state=42)
forest_model.fit(x_train, y_train)
feature_importances = list(zip(x_train.columns, forest_model.feature_importances_))
for feature, importance in feature_importances:
    print(f'{feature}: {importance}')

AaAno: 0.1936452943756041
MmMes: 0.3939393300172057
DdDia: 0.024948546464241744
VzQJusante: 0.11076956766671252
VzQNatural: 0.1603806450850261
VrChuva: 0.0028652956489682237
VrTempMin: 0.028967705137606645
VrTempMed: 0.01156134376914812
VrTempMax: 0.014164213116936917
VrUmidadeArMin: 0.00925520924346177
VrUmidadeArMed: 0.01029797238643453
VrUmidadeArMax: 0.03620856217172634
VrPrecipitacao: 0.0029963149169273174

```

Fonte: Elaborada pelos autores (2023).

Figura 18: Valores do RMSE e do R<sup>2</sup>

```
rmse1 = np.sqrt(mean_squared_error(y1_test, y1_pred))
#modelo2
print(f'rmse:{rmse1}, r2:{r21}')
[731] ✓ 0.0s
...
rmse:0.7329208001903514, r2:0.8986779278706971
rmse:0.6424826606212609, r2:0.9221402882545311
```

Fonte: Elaborada pelos autores (2023).

A avaliação do modelo foi feita usando as mesmas métricas utilizadas para o *Decision Tree Regressor*. Os resultados do modelo *Random Forest Regressor* foram os seguintes:

RMSE: 0,642  
R<sup>2</sup>: 0,922

- *XGBoost*

O *XGBoost* (*Extreme Gradient Boosting*) é um algoritmo de *boosting* que visa melhorar o desempenho de modelos fracos (como árvores de decisão), combinando-os de maneira sequencial. O funcionamento do modelo é criar uma sequência de modelos, onde cada modelo tenta corrigir os erros do anterior. A avaliação do modelo foi feita usando as mesmas métricas utilizadas para os modelos anteriores. Os resultados do modelo *XGBoost* foram os seguintes:

RMSE: 0.871  
R<sup>2</sup>: 0,856

Figura 19 - Modelo XGBoost

```

# Criar um objeto DMatrix
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# Parâmetros do modelo
params = {
    'max_depth': 3,
    'learning_rate': 0.1,
    'objective': 'reg:squarederror', # Problema de regressão
}

# Treinar o modelo
num_round = 100
model = xgb.train(params, dtrain, num_round)

predictions = model.predict(dtest)

# Calcular o RMSE
rmse = np.sqrt(mean_squared_error(y_test, predictions))
r2 = r2_score(y_test, predictions)
print(f'rmse:{rmse}, r2:{r2}')

rmse:0.8715327736396948, r2:0.8567292893082376

```

Fonte: Elaborada pelos autores (2023).

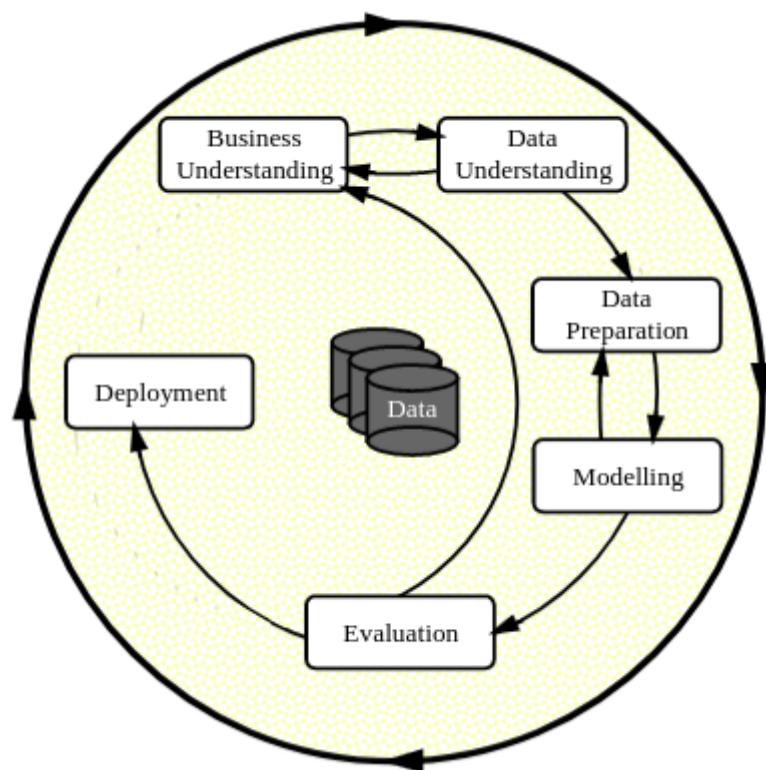
Desta forma, os resultados dos modelos não-paramétricos foram semelhantes, com o *Random Forest Regressor* apresentando o melhor desempenho, com um RMSE ligeiramente inferior. No entanto, todos os modelos apresentaram um bom desempenho, com um R<sup>2</sup> próximo de 1.

A escolha do melhor modelo deve ser feita levando em consideração fatores como a complexidade do modelo, o tempo de treinamento e a facilidade de interpretação. No caso deste estudo, o *Random Forest Regressor* foi escolhido por apresentar o melhor desempenho e ser relativamente simples de interpretar.

## 5.4 Construção do *dashboard*

Dentre as metodologias usadas em projetos de dados, como a SEMMA e o KDD, optamos pelo CRISP-DM. O CRISP-DM é um modelo para processos de mineração de dados e descoberta de conhecimento que contempla seis etapas principais: entendimento do negócio, entendimento dos dados, preparação dos dados, modelagem, avaliação e implementação, conforme ilustrado na figura 20 (WIRTH; HIPP, 2000).

**Figura 20 – Etapas do CRISP-DM**



Fonte: Wirth e Hipp (2000).

No nosso trabalho, a fase de entendimento do negócio do CRISP-DM coincidiu com a etapa de inspiração do *design thinking*: buscamos entender o contexto, procurar trabalhos relacionados e elaborar requisitos. Na fase de entendimento dos dados, consultamos o dicionário de dados disponibilizado pela SABESP que descreve as colunas e pensamos na melhor forma de exibir esses dados.

A preparação dos dados envolveu a transformação dos dados em um formato mais conveniente. A modelagem, no nosso contexto, foi a elaboração do dashboard a

partir do refinamento do protótipo construído anteriormente. A avaliação foi realizada através de um questionário destinado a habitantes da Região Metropolitana de São Paulo. Por fim, o *dashboard* foi implementado através do *framework Streamlit*<sup>6</sup>, que permite a criação de aplicações *web* voltadas à visualização de dados de forma simples através da linguagem *Python*, e será atualizado diariamente com os dados fornecidos pela API. O código-fonte está disponível em um repositório no GitHub<sup>7</sup>.

## 5.5 Construção da API

Uma API (Interface de Programação de Aplicação, do inglês *Application Programming Interface*) é uma interface entre duas aplicações, permitindo o uso de recursos e transferência de dados através de um protocolo de comunicação definido. Quando esses dados são fornecidos via *web* por uma aplicação autônoma, definimos essa aplicação como um *web service*. Há vários padrões e tecnologias de *web services*, como SOAP, CORBA, DCOM e RMI, mas a tecnologia mais usada no mundo é o modelo REST, também conhecido como RESTful (PEREIRA *et. al.*, 2018).

Segundo Pereira *et. al* (2018), o modelo REST identifica recursos através de um nome definido (normalmente uma ID), usa operadores de métodos padronizados através do protocolo HTTP e representa recursos em formatos como XML e JSON. Trata-se de uma arquitetura de *web service* simples que abstrai a arquitetura da *World Wide Web*, o que contribui para sua popularidade.

No nosso trabalho, construímos um *web service* com sua respectiva API REST através do framework *Flask*<sup>8</sup> da linguagem *Python*, escolhido devido à simplicidade da ferramenta e familiaridade dos membros com a linguagem. O código-fonte está disponível em um repositório no GitHub<sup>9</sup>.

---

<sup>6</sup> <https://streamlit.io/>

<sup>7</sup> <https://github.com/GMerencio/sabesp-dashboard>

<sup>8</sup> <https://flask.palletsprojects.com/en/2.3.x/>

<sup>9</sup> <https://github.com/GMerencio/sabesp-api>

Com relação aos recursos e custos, hospedamos a API na plataforma *cloud* Render<sup>10</sup>, que disponibiliza um plano gratuito que atende as necessidades do trabalho. Caso o plano seja cancelado ou alterado, podemos procurar um serviço alternativo. Adicionalmente, como o projeto é de código aberto, desenvolvedores interessados no projeto hospedar a API em outras plataformas.

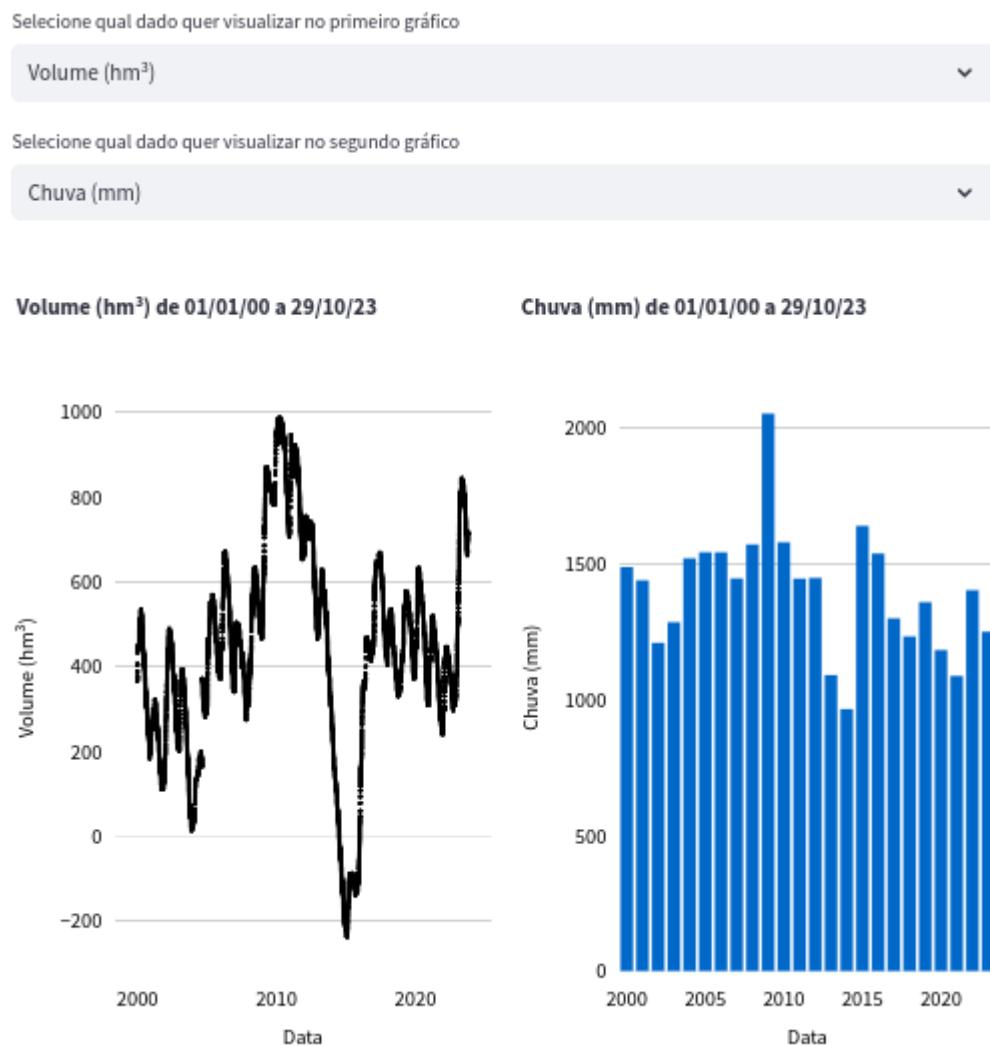
---

<sup>10</sup> <https://render.com/>

## 6 Resultados e discussões

A partir do protótipo do *dashboard*, implementamos as funcionalidades e refinamos o *design* de acordo com as discussões internas, publicando a versão final da aplicação<sup>11</sup>. Em particular, seguindo as sugestões da orientadora, optamos por usar um gráfico de barras em vez de um gráfico de dispersão para os dados de chuva, facilitando a visualização, e incluímos a opção de visualizar dois gráficos ao mesmo tempo para permitir a comparação entre variáveis em um determinado período de tempo, como exibido na figura 21.

**Figura 21 – Dashboard final após melhorias**



Fonte: Elaborada pelos autores (2023).

<sup>11</sup> <https://sabesp-dashboard.streamlit.app/>

Também finalizamos a página “Sobre os dados”, contendo informações básicas sobre a fonte dos dados e o que cada coluna representa (figura 22) e a página “API”, que direciona o usuário à página de documentação e repositório do GitHub da API (figura 23).

**Figura 22** – Página “Sobre os dados” na aplicação final

## Sobre os dados

A fonte dos dados do projeto é a [Sabesp](#), que disponibiliza dados referentes ao sistemas produtores. A nível de sistemas, as seguintes informações estão disponíveis:

- Data: data do registro, no formato AAAA-MM-DD (por exemplo, 2000-12-31);
- Volume ( $hm^3$ ): corresponde ao volume armazenado, em hectômetros cúbicos;
- Vazão natural ( $m^3/s$ ): representa a vazão média diária afluente, em metros cúbicos por segundo, sem considerar as alterações antrópicas;
- Vazão a jusante ( $m^3/s$ ): indica a vazão média diária descarregada, em metros cúbicos por segundo, para a jusante da barragem, ou seja, a quantidade de água liberada que segue pelo rio ou canal após o barramento;
- Chuva (mm): refere-se à precipitação acumulada nas últimas 24 horas no local do barramento, em milímetros.

Os dados são atualizados diariamente através de um script e disponibilizados via API. Você pode acessar o código-fonte e os dados, incluindo os dados dos reservatórios de cada sistema, [neste repositório do GitHub](#).

Fonte: Elaborada pelos autores (2023).

**Figura 23** – Página “API” na aplicação final

## API

Desenvolvedores podem acessar a [API REST](#) que disponibiliza os dados de sistemas e reservatórios, atualizados diariamente. [O repositório do projeto pode ser acessado aqui](#).

Fonte: Elaborada pelos autores (2023).

Paralelamente ao desenvolvimento do *dashboard*, finalizamos a API e a disponibilizamos através da plataforma *cloud Render*<sup>12</sup>. A API fornece quatro endpoints, conforme exibido na figura 24:

- **/sistemas/{inicio}&{fim}&{sistema}** (GET): Fornece os dados de um determinado sistema no período especificado;
- **/sistemas/{sistema}** (GET): Fornece os dados de hoje de um determinado sistema;
- **/reservatorios/{inicio}&{fim}&{sistema}** (GET): Fornece os dados dos reservatórios de um determinado sistema no período especificado;
- **/reservatorios/{sistema}** (GET): Fornece os dados de hoje dos reservatórios de um determinado sistema.

**Figura 24 – Endpoints da API na documentação**

The screenshot shows the API SABESP 1.0 documentation. At the top, it says "API SABESP 1.0" and "[ Base URL: / ] /swagger.json". Below that, a note states: "API que disponibiliza os dados de reservatórios e sistemas da SABESP, atualizados diariamente. A verificação de novos dados ocorre, aproximadamente, às 09:00, 10:00, 15:00 e 20:00 no horário de Brasília." The documentation is organized into sections: "sistemas" and "reservatorios". Under "sistemas", there are two GET endpoints: one for "/sistemas/{inicio}&{fim}&{sistema}" (Dados do sistema no período especificado) and another for "/sistemas/{sistema}" (Dados de hoje do sistema especificado, caso haja). Under "reservatorios", there are also two GET endpoints: one for "/reservatorios/{inicio}&{fim}&{sistema}" (Dados dos reservatórios do sistema no período especificado) and another for "/reservatorios/{sistema}" (Dados de hoje dos reservatórios do sistema especificado, caso haja).

Fonte: Elaborada pelos autores (2023).

A documentação da API descreve cada *endpoint*, com os parâmetros esperados e respostas (conforme a figura 25), e permite o teste de cada um deles (conforme a figura 26).

<sup>12</sup> <https://sabesp-api.onrender.com/>

**Figura 25 – Documentação de um endpoint da API**

**GET /sistemas/{sistema}** Dados de hoje do sistema especificado, caso haja

**Parameters**

Name	Description
sistema <small>* required string (path)</small>	Nome do sistema. Opções: cantareira, alto_tiete, rio_claro, rio_grande, guarapiranga, cotia, sao_lourenco

**Responses**

Code	Description	Response content type
200	Success	application/json
400	Solicitação inválida. Verifique a sintaxe da requisição	
404	Os dados de hoje ainda não foram atualizados	

Example Value Model

```
{
  "Data": "2000-12-31",
  "Volume (hm³)": 120,
  "Volume (%)": 50.5,
  "Chuva (mm)": 10,
  "Vazão natural (m³/s)": 20.8,
  "Vazão a jusante (m³/s)": 17.2
}
```

Fonte: Elaborada pelos autores (2023).

**Figura 26 – Resultado de um teste de um endpoint da API**

**Responses**

Code	Details	Response content type
200	Response body	application/json

Curl

```
curl -X 'GET' \
  'https://sabesp-api.onrender.com/sistemas/cantareira' \
  -H 'accept: application/json'
```

Request URL

```
https://sabesp-api.onrender.com/sistemas/cantareira
```

Server response

Code	Details
200	Response body

```
[
  {
    "Data": "2023-10-29",
    "Volume (hm³)": 712.52388,
    "Volume (%)": 72.55312,
    "Chuva (mm)": 0,
    "Vazão natural (m³/s)": 44.593,
    "Vazão a jusante (m³/s)": 8.35
  }
]
```

Response headers

```
alt-svc: h3="443"; ma=96400
cf-cache-status: DYNAMIC
cf-ray: 81dd162a0bheaf0-GRU
content-encoding: br
content-type: application/json
date: Sun, 29 Oct 2023 17:19:59 GMT
rndr-id: 63cbfedf-4bf4-4023
server: cloudflare
vary: Accept-Encoding
x-firefox-early-data: accepted
x-firefox-htp3: h3
x-render-origin-server: unicorn
```

Fonte: Elaborada pelos autores (2023).

A documentação da API também traz informações dos dados, incluindo os tipos e descrições de cada campo, como exibe a figura 27.

**Figura 27** – Descrição dos dados fornecidos pela API na documentação

Models	
<b>Sistema</b> ↴ {	
Data	string(\$date) example: 2000-12-31  Data do registro
Volume (hm <sup>3</sup> )	number example: 120  Volume armazenado, em hectômetros cúbicos
Volume (%)	number example: 50.5  Volume armazenado, em percentual do volume total
Chuva (mm)	number example: 10  Precipitação acumulada das últimas 24 horas, em milímetros
Vazão natural (m <sup>3</sup> /s)	number example: 20.8  Vazão média diária afluente, em metros cúbicos por segundo
Vazão a jusante (m <sup>3</sup> /s)	number example: 17.2  Vazão média diária descarregada, em metros cúbicos por segundo
}	
<b>Reservatório</b> ↴ {	
Data	string(\$date) example: 2000-12-31  Data do registro

Fonte: Elaborada pelos autores (2023).

Com a API e o *dashboard* finalizados, criamos uma pesquisa de avaliação da aplicação através do *Google Forms* e a distribuímos entre 20/10 e 25/10 por meio de grupos de *WhatsApp*, *Telegram*, redes sociais e outros meios de comunicação, focando no público residente na Região Metropolitana de São Paulo. Fizemos as seguintes perguntas:

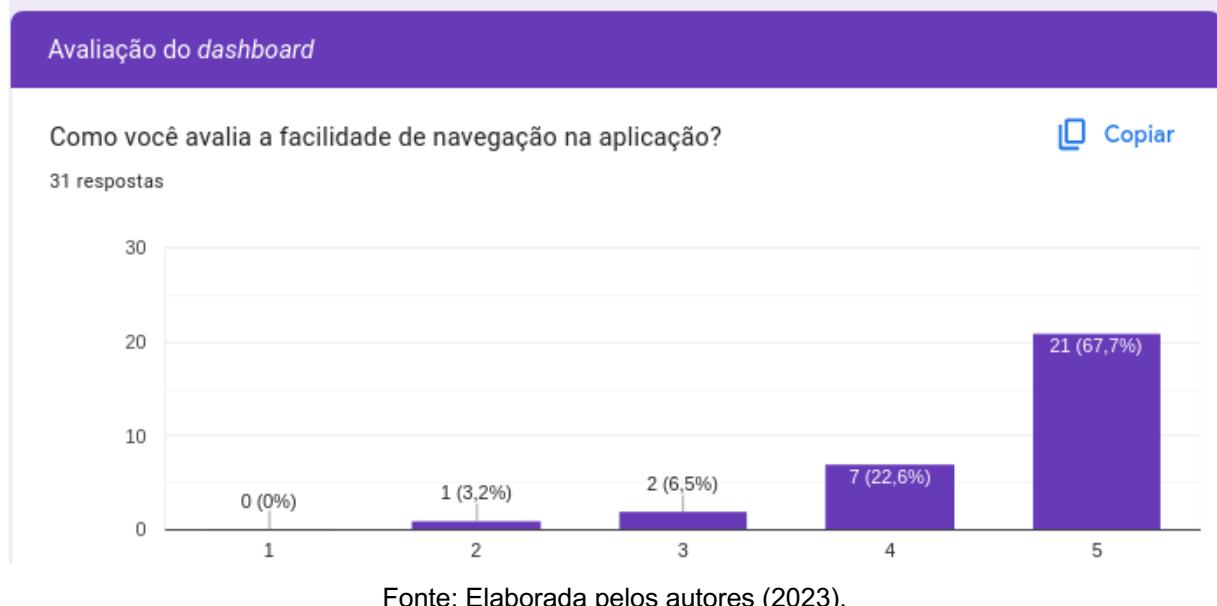
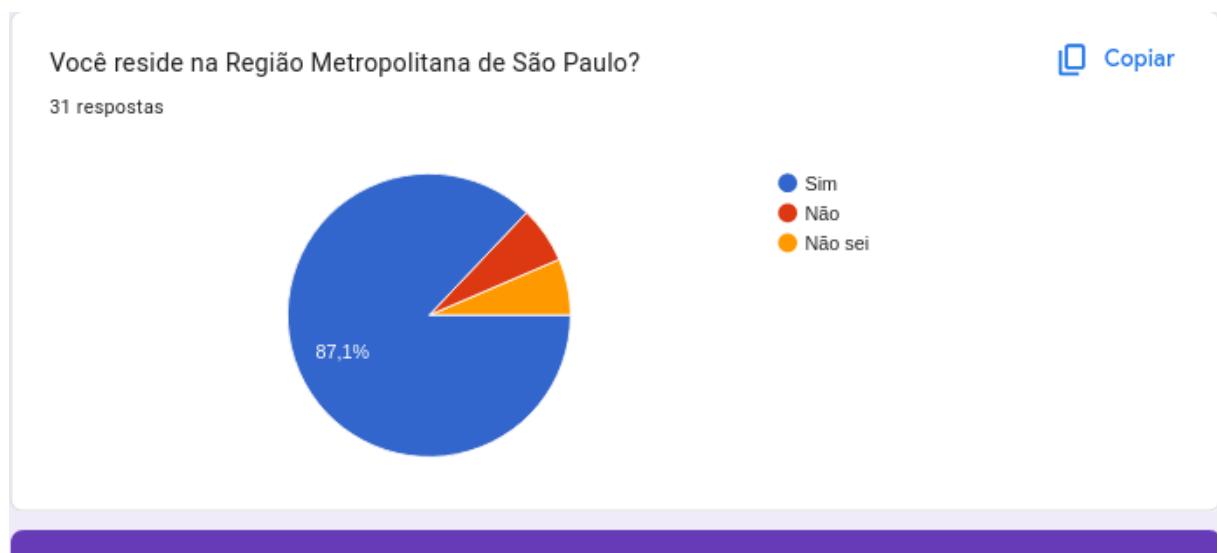
1. Você reside na Região Metropolitana de São Paulo?
  2. Como você avalia a facilidade de navegação na aplicação?
  3. (Opcional) Por que você deu essa nota à facilidade de navegação na aplicação?
  4. Como você avalia a facilidade de interagir com os gráficos (selecionando o período, o dado a ser exibido, etc.)?
  5. (Opcional) Por que você deu essa nota à facilidade de interagir com os gráficos?
  6. Como você avalia a velocidade da aplicação (tempo que as páginas demoram para carregarem e atualizarem)?
  7. O quanto útil você considera a aplicação?
  8. Por favor justifique sua avaliação da utilidade da aplicação.
  9. Você encontrou algum erro ou bug ao navegar pela aplicação?

- 10.(Opcional) Caso tenha encontrado algum erro ou bug ao navegar pela aplicação, por favor descreva-o(s) aqui.
- 11.(Opcional) Caso tenha algum comentário adicional, por favor insira aqui.

Coletamos um total de 31 respostas. De modo geral, vimos um *feedback* positivo em relação à facilidade de navegação, facilidade de interação com os gráficos, velocidade da aplicação e utilidade. As figuras 28 a 33 mostram o resumo das respostas. Analisando as respostas, notamos os seguintes pontos:

- Alguns usuários notaram dificuldades em acessar a aplicação pelo celular;
- Tentamos reproduzir os *bugs* relatados, sem êxito. Vale ressaltar que o *Streamlit* ocasionalmente exibe mensagens de erro que não afetam a aplicação, bem como comportamentos inesperados temporários;
- Alguns usuários indicaram dificuldade em navegar pelo gráfico, inclusive sugerindo instruções;
- Outras sugestões incluem uma funcionalidade que indique qual sistema abastece o local de residência do usuário, um aplicativo móvel e notificações de volume baixo nos sistemas.

**Figura 28** – Resumo das respostas às perguntas 1 e 2 do questionário de avaliação



Fonte: Elaborada pelos autores (2023).

**Figura 29** – Resumo das respostas às perguntas 3 e 4 do questionário de avaliação

(Opcional) Por que você deu essa nota à facilidade de navegação na aplicação?

3 respostas

Layout simples, intuitivo

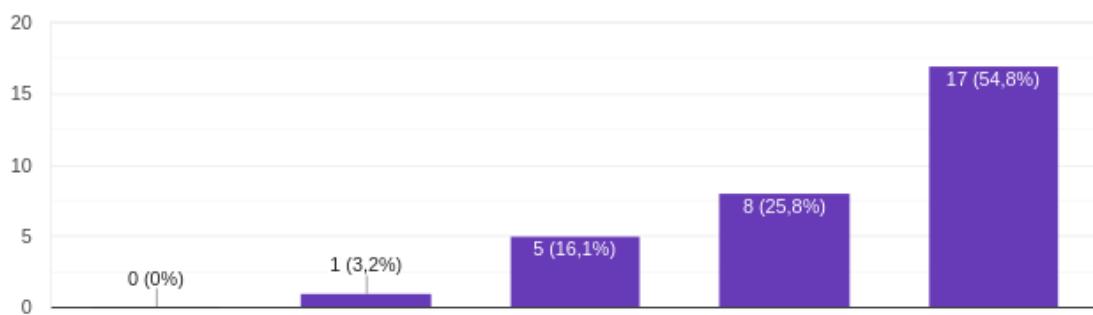
Instruções claras

Não achei fácil de usar no celular

Como você avalia a facilidade de interagir com os gráficos (selecionando o período, o dado a ser exibido, etc.)?

 Copiar

31 respostas



Fonte: Elaborada pelos autores (2023).

**Figura 30–** Resumo das respostas às perguntas 4 e 5 do questionário de avaliação

(Opcional) Por que você deu essa nota à facilidade de interagir com os gráficos?

5 respostas

Poderia ter instruções.

Os filtros são intuitivos e funcionam como esperado, mas não entendi como interagir com os gráficos clicando neles

Fácil e rápido pra escolher os dados

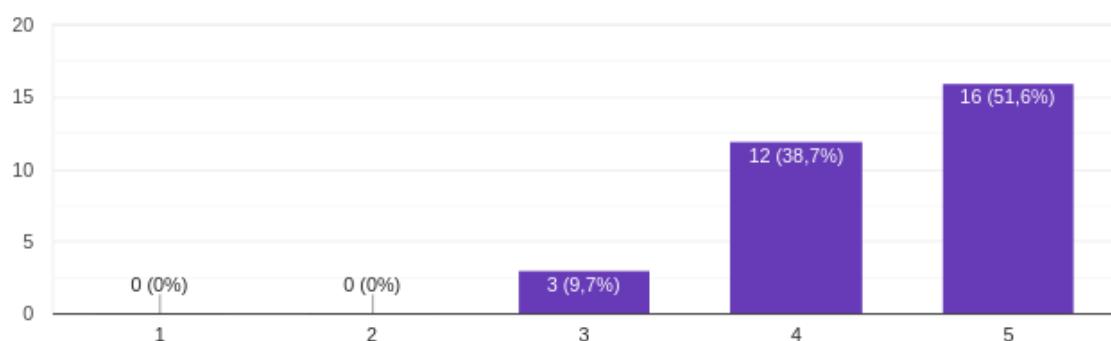
nao sei mexer com gráficos

Ruim de mexer no celular

Como você avalia a velocidade da aplicação (tempo que as páginas demoram para carregarem e atualizarem)?

Copiar

31 respostas



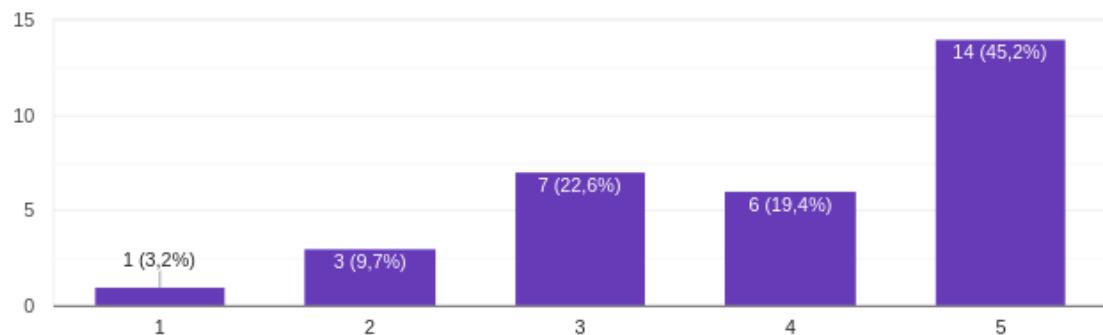
Fonte: Elaborada pelos autores (2023).

**Figura 31** – Resumo das respostas às perguntas 6 e 7 do questionário de avaliação

O quanto útil você considera a aplicação?

 Copiar

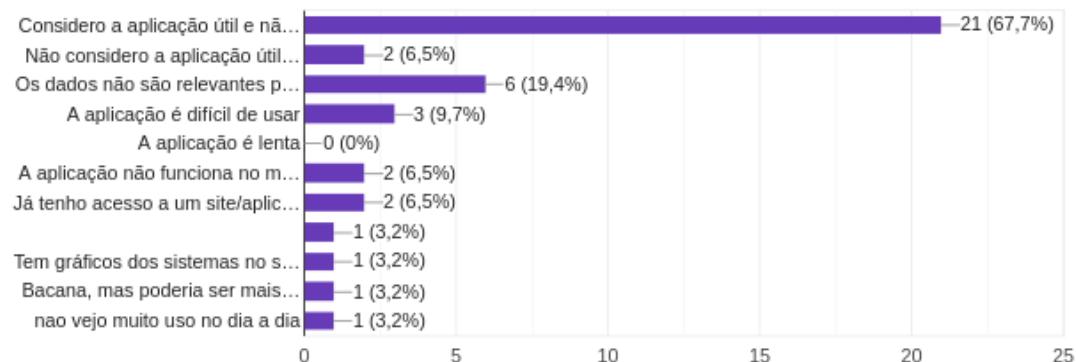
31 respostas



Por favor justifique sua avaliação da utilidade da aplicação.

 Copiar

31 respostas



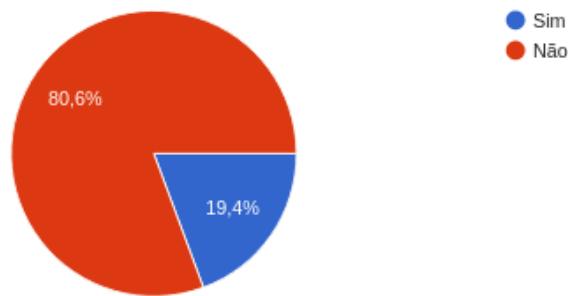
Fonte: Elaborada pelos autores (2023).

**Figura 32** – Resumo das respostas às perguntas 8 e 9 do questionário de avaliação

Você encontrou algum erro ou bug ao navegar pela aplicação?

 Copiar

31 respostas



(Opcional) Caso tenha encontrado algum erro ou bug ao navegar pela aplicação, por favor descreva-o(s) aqui.

3 respostas

Mensagem "Bad message format: s is undefined"

site n carregou no celular

Às vezes o gráfico não muda quando vai de uma página pra outra

Fonte: Elaborada pelos autores (2023).

**Figura 33 – Resumo das respostas à pergunta 10 do questionário de avaliação**

(Opcional) Caso tenha algum comentário adicional, por favor insira aqui.

4 respostas

não sei que sistema abastece minha região... seria bom ter algo que mostrasse isso

Explicar como funciona a interatividade com os gráficos e talvez bolar um sistema de notificação de quando o nível do reservatório está baixo, etc.

Muito bom, mas não moro em São Paulo

Poderia ter um app

Fonte: Elaborada pelos autores (2023).

## 7 Conclusões

O desenvolvimento desse trabalho de conclusão de curso teve como base o acesso facilitado e transparente a dados e informações sobre recursos hídricos, assunto de suma importância à sociedade. Diante disso, foram selecionados os dados disponibilizados das represas administradas pela SABESP (Companhia de Saneamento Básico de São Paulo), responsável pelo abastecimento de água do estado de São Paulo.

Analizando outros projetos com propósitos em comum, foram encontradas algumas deficiências, como a falta de disponibilização pública das informações trabalhadas, a ausência de expansão dos dados abrangendo mais represas e mais períodos, versão *web* para consulta, delimitando o acesso a apenas dispositivos móveis, inexistência da possibilidade da extração de dados atualizados do site e acesso limitado apenas a uma estrutura de programação.

Com base nos aspectos levantados e identificação dessas lacunas, e levando em conta interesses da população e os dados encontrados no site da empresa SABESP, foram determinados dois objetivos: o primeiro foi construir uma base de acesso atualizada com os dados através de uma API para pesquisadores, desenvolvedores, técnicos e curiosos sobre o assunto. E o segundo foi a elaboração de um *dashboard* dinâmico e interativo para o público em geral.

Partindo da metodologia de *design thinking*, foram definidas as etapas, iniciando pela escolha das ferramentas para extração e tratamento de dados. Os dados foram coletados através de *web scraping* com a linguagem *Python*, usando a plataforma Github Actions para automatizar a coleta, e o tratamento de dados foi realizado utilizando a biblioteca *Pandas*. Para a API, a arquitetura REST foi selecionada, resultando em um *web service* desenvolvido com o *framework Flask* em *Python*. O *dashboard* foi estruturado empregando a estrutura de modelagem CRISP-DM e implementado através da biblioteca *Streamlit*.

Executamos os objetivos que foram propostos com a criação de um ambiente completo, com acesso a todo material disponibilizado pela SABESP sobre todas as suas represas e sistemas. Através da combinação da extração automatizada de dados, fornecimento dos dados através da API e visualização por meio do *dashboard*, promovemos a ampliação do acesso público à informação de maneira mais contextualizada, focada, dinâmica e clara. Verificamos a satisfação da sociedade através do questionário aplicado, constatando um retorno positivo e interesse da população.

A API, acessível através de métodos HTTP, pode ser usada em qualquer linguagem de programação. Além disso, com os dados atualizados e disponibilizados em um formato padrão, pesquisadores, desenvolvedores e outras partes interessadas nesses dados podem trabalhar apenas na modelagem e no desenvolvimento de soluções, em vez de se preocuparem excessivamente com a extração e tratamento dos dados, o que contribui para a facilitação de futuros projetos e ideias.

Por ser uma aplicação *web*, o *dashboard* é acessível por qualquer tipo de dispositivo. O *dashboard* também oferece opções interativas para comparar variáveis e períodos, trazendo melhorias na forma de visualização desses dados por parte da população em geral em comparação ao formato tabular fornecido pelo portal da SABESP. Os gráficos permitem atrelar os dados a possíveis causas ou impactos, como por exemplo a associação de baixos volumes em reservatórios a fenômenos naturais, como o baixo volume de chuva.

Perante o desenvolvido, o projeto pode vir a contribuir para a gestão de recursos hídricos com a participação da sociedade. Tratando-se de um projeto de código aberto, há a possibilidade de melhorias pela comunidade, como a expansão de dados coletados, desenvolvimento de modelos preditivos e aprimoramento do *dashboard* a partir do *feedback* colhido.

## 8 Referências

ALMEIDA, A. S. et al. **Sistema de previsão hidrometeorológico para subsidiar a operação do sistema Cantareira na gestão das bacias PCJ.** In: Simpósio Brasileiro de Recursos Hídricos - SBRH, 23, 2019, Foz do Iguaçu, PR. Anais eletrônicos... Florianópolis, SC: ABRHidro, 2019. Disponível em: <<https://files.abrhidro.org.br/Eventos/Trabalhos/107/XXIII-SBRH0640-1-20190812-223215.pdf>>. Acesso em: 29 ago. 2023.

BEZERRA, F. **Controle Social, Democracia e Administração Pública.** Controladoria Geral da União - CGU, fev. 2021, Brasília. Disponível em: <<https://www.gov.br/cgu/pt-br/assuntos/controle-social/artigos/controle-social-democracia-e-administracao-publica>>. Acesso em 13 de set. 2023.

BRASIL. **Lei Nº12.527** de 18 de nov. de 2011. Regula o acesso a informações previsto no inciso XXXIII do art. 5º , no inciso II do § 3º do art. 37 e no § 2º do art. 216 da Constituição Federal; altera a Lei nº 8.112, de 11 de dezembro de 1990; revoga a Lei nº 11.111, de 5 de maio de 2005, e dispositivos da Lei nº 8.159, de 8 de janeiro de 1991; e dá outras providências. Diário Oficial da União, Brasília, 18 nov. de 2011. Disponível em: <[https://www.planalto.gov.br/ccivil\\_03/ato2011-2014/lei/l12527.htm](https://www.planalto.gov.br/ccivil_03/ato2011-2014/lei/l12527.htm)>. Acesso em 16 set. 2023.

BROWN, T.; WYATT, J. **Design thinking for Social Innovation.** Stanford Social Innovation Review. California: Leland Stanford Jr. University, 2010.

CHEN, D. Y. **Análise de dados com Python e Pandas.** Tradução de Lúcia A. Kinoshita. 1ª ed. São Paulo: Novatec editora, 2018

COSTA, S. E. da. **Preparação e Análise Exploratória de Dados.** Indaiatuba: Uniasselvi, 2020.

DEZEM, V. **Falta d'água paralisa fábricas e ameaça o crescimento da economia de São Paulo.** UOL Economia, out. 2014. Disponível em: <<https://economia.uol.com.br/noticias/bloomberg/2014/10/24/falta-dagua-paralisa-unidades-da-rhodia-e-afeta-negocios-em-sp.htm>>. Acesso em: 29 ago. 2023.

MCKINNEY, W. **Python para Análise de dados:** Tratamento de dados com pandas, numpy e ipython. Tradução de Lúcia A. Kinoshita. 1ª ed. São Paulo: Novatec, 2017.

MILZ, B. **beatrizmilz/mananciais:** Base de dados sobre volume operacional em mananciais de abastecimento público na Região Metropolitana de São Paulo (SP - Brasil). Disponível em: <<https://github.com/beatrizmilz/mananciais>>. Acesso em: 29 ago. 2023.

MIOT, H. A. (2017). **Avaliação da normalidade dos dados em estudos clínicos e experimentais**. Porto Alegre: Jornal Vascular Brasileiro, 2007. p88-p91.

MITCHELL, R. **Web Scraping com Python**. São Paulo: Novatec Editora, 2019.

PEREIRA, A. L K. et al. **Abordagem Restful Em Serviços Web**. Revista Gestão em Foco, n. 10, p. 460-465, 2018. Disponível em: <<https://portal.unisepe.com.br/unifia/wp-content/uploads/sites/10001/2018/12/032-ABORDAGEM-RESTFUL-EM-SERVI%C3%87OS-WEB.pdf>>. Acesso em: 29 ago. 2023.

PRESBITERIS, R. **Obtendo atualizações dos reservatórios de água da SABESP**. Portal iMasters, abr. 2015. Disponível em: <<https://imasters.com.br/apis-microsservicos/obtendo-atualizacoes-dos-reservatorios-de-agua-da-sabesp>>. Acesso em: 29 ago. 2023.

REDAÇÃO RBA. **Falta de água em São Paulo já causa 3 mil demissões**. Rede Brasil Atual, jul. 2014. Disponível em: <<https://www.redebrasilitual.com.br/cidadania/falta-de-agua-em-sao-paulo-ja-causa-3-mil-demissoes-8539/>>. Acesso em: 29 ago. 2023.

SABESP. s.d. **De Onde Vem?**. Disponível em: <<https://site.sabesp.com.br/site/interna/Default.aspx?secaold=31>>. Acesso em 16 set. 2023.

SOUZA-FERNANDES, L. C. **A escassez de água no Sistema Cantareira**. In: SADDY, A.; BRANDÃO, C.; AVZARADEL, P. C. S. (Orgs.). **Constituição, Crise Hídrica, Energia e Mineração na América Latina**. Rio de Janeiro: Lumen Juris, 2016. p. 89-117.

WIRTH, R.; Hipp, J. **CRISP-DM: towards a standard process model for data mining**. In: International Conference and Exhibition on the Practical Application of Knowledge Discovery and Data Mining, 4, 2000, Manchester, UK. **Anais...** Blackpool, UK: Practical Application Company, 2000, p. 29-40.

VIEIRA, B. S. et. al. **Previsão de volume do Sistema Cantareira com modelos de aprendizado de máquina**. 61f. Relatório Técnico-Científico. Cursos de Ciência de Dados e Engenharia da Computação – **Universidade Virtual do Estado de São Paulo**. Tutor: Nícolas Rosalem. Polos CEU Jambeiro, Quinta do Sol, Alto Alegre, Feitiço da Vila, Parque Bristol, Capão Redondo e Parque São Carlos, 2023.

## 9 Apêndices

### APÊNDICE A - TRATAMENTO DE DADOS

#### Lista de figuras - Apêndice A

Figura A.1 – Função de tratamento dos arquivos exportados em formato xml.....	47
Figura A.2 – Verificação do formato CSV do arquivo tratado.....	48
Figura A.3 – Concatenação dos <i>Dataframes</i> e ordenação do <i>DataFrame</i> resultante.....	48
Figura A.4 – Verificação dos tipos do <i>DataFrame</i> .....	49
Figura A.5 – Conversão da coluna ‘Data’.....	50
Figura A.6 – Contagem dos valores nulos.....	51
Figura A.7 – Verificação de duplicidade nos nomes e valores das colunas.....	52
Figura A.8 – Verificação e remoção de duplicidade.....	53
Figura A.9 – Verificação de <i>outliers</i> .....	53

Conforme ilustrado na figura 2, o arquivo exportado está em formato XML e necessitará realizar um tratamento para que possamos utilizá-los posteriormente. Desta forma, foi realizado o *parsing* do documento *xml* para *Dataframe*:

**Figura A.1** – Função de tratamento dos arquivos exportados em formato xml

```

1 def parse_xml(filepath):
2     df = None
3     with open(filepath,'r') as xml_file:
4         soup = BeautifulSoup(xml_file.read(), 'xml')
5         for sheet in soup.findAll('Worksheet'):
6             sheet_as_list = []
7             for row in sheet.findAll('Row'):
8                 sheet_as_list.append([cell.Data.text if cell.Data else '' for cell in row.findAll('Cell')])
9             df = pd.DataFrame(sheet_as_list)
10            df.columns = df.iloc[1]
11            df.drop([0, 1], inplace=True)
12            df.reset_index(drop=True, inplace=True)
13            df['Data'] = pd.to_datetime(df['Data'], dayfirst=True)
14    return df

```

Fonte: Elaborada pelos autores (2023)

Após o tratamento realizado na figura A.1, obtivemos um arquivo em formato CSV, como demonstrado na figura A.2.

**Figura A.2 – Verificação do formato CSV do arquivo tratado**

amostra.csv
1 Data,Represa,Nível (m),Volume (hm³),Volume (%),Q Jusante (m³/s),Q Natural (m³/s),Chuva (mm)
2 2023-04-01,Jaguari/Jacareí,842.73,745.9688920815145,92.31790709462861,0.25,21.33237776593635,0
3 2023-04-01,Cachoeira,817.48,34.799797145691024,49.964276510791905,0.5,3.6913948468352515,11
4 2023-04-01,Atibainha,782.92,19.12603995065382,19.870760289778314,1,5.332554589490428,11.2
5 2023-04-01,Paiva Castro,744.13,1.2915293135742445,16.966837071325614,0.1,5.697079795496492,13
6 2023-04-02,Jaguari/Jacareí,842.73,745.9688920815145,92.31790709462861,0.25,31.25,0
7 2023-04-02,Cachoeira,817.53,35.15359675558181,50.47224905626355,0.5,2.3889028922544613,0.2
8 2023-04-02,Atibainha,782.98,20.25533066764399,21.04402277342258,1,5.962494409608443,0.4
9 2023-04-02,Paiva Castro,744.11,1.212089003448309,15.923228703605167,0.1,3.313551966135004,0.2
10 2023-04-03,Jaguari/Jacareí,842.72,745.4879739395055,92.25839073030647,0.25,25.18381780082255,0
11 2023-04-03,Cachoeira,817.57,35.43735617791514,50.87966046100965,0.5,2.115252573302463,0
12 2023-04-03,Atibainha,783.02,21.009755445188034,21.827822971996074,1,1.5307682586116336,0
13 2023-04-03,Paiva Castro,744.12,1.2517901789658765,16.444783552935306,0.1,3.2245043462681444,0
14 2023-04-04,Jaguari/Jacareí,842.71,745.00718025165,92.19888976789599,0.25,25.185258242412424,0
15 2023-04-04,Cachoeira,817.6,35.65059606869297,51.18582250044192,0.5,1.3480542914100466,0
16 2023-04-04,Atibainha,783.08,22.143738665180933,23.005960673017636,1,3.9888057869548526,0
17 2023-04-04,Paiva Castro,744.12,1.2517901789658765,16.444783552935306,0.1,3.5989999999999999,0
18 2023-04-05,Jaguari/Jacareí,842.7,744.5265110539269,92.13940421184978,0.25,24.886699100427712,0

Fonte: Elaborada pelos autores (2023).

Tendo apenas a possibilidade de se coletar os dados por ano de cada represa, foi preciso realizar a junção dos *Dataframes* com a função *concat* (figura A.3).

**Figura A.3** – Concatenação dos *Dataframes* e ordenação do *DataFrame* resultante

```
1 df_final = pd.concat([df1, df2, df3])
2 df_final.sort_values(by=['Data'], inplace=True)
3 df_final
```

1	Data	Volume (hm³)	Volume (%)	Chuva (mm)	Vazão natural (m³/s)	Vazão a jusante (m³/s)
<b>1460</b>	2011-01-01	735.97625	74.94117	0.1	30.346	5.5
<b>1459</b>	2011-01-02	739.48751	75.29871	36.25	70.475	5.5
<b>1458</b>	2011-01-03	749.68231	76.3368	50.1	147.125	4.48
<b>1457</b>	2011-01-04	774.73991	78.8883	43.95	321.239	2.27
<b>1456</b>	2011-01-05	791.33379	80.57798	8.2	223.21	2
...	...	...	...	...	...	...
<b>4</b>	2022-12-28	391.29407	39.84373	29.7	60.422	1.17
<b>3</b>	2022-12-29	399.4439	40.67359	24.5	114.401	0.85
<b>2</b>	2022-12-30	407.63398	41.50754	38.6	115.444	0.85
<b>1</b>	2022-12-31	414.72548	42.22964	12.7	104.118	0.85
<b>0</b>	2023-01-01	420.23007	42.79015	10.05	86.009	0.85

4384 rows × 6 columns

Fonte: Elaborada pelos autores (2023).

Após a concatenação, o passo seguinte a ser realizado foi verificar a consistência e integridade dos dados e realização de tratamentos adicionais, caso preciso. Para isso, foi utilizado a função *info*, conforme figura A.4.

**Figura A.4** – Verificação dos tipos do *DataFrame*

## ▼ Verificação dos tipos

```
[ ] df_alto_tiete.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8684 entries, 0 to 8683
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype  
---  --  
 0   Data              8684 non-null    object 
 1   Volume (hm³)      8684 non-null    float64
 2   Volume (%)        8684 non-null    float64
 3   Chuva (mm)       6857 non-null    float64
 4   Vazão natural (m³/s) 8684 non-null    float64
 5   Vazão a jusante (m³/s) 8684 non-null    float64
dtypes: float64(5), object(1)
memory usage: 407.2+ KB
```

Fazer conversão das colunas de data e verificar novamente para todos os DataFrames.

Fonte: Elaborada pelos autores (2023).

Como a coluna Data apresentou um formato inadequado à sua manipulação, foi preciso realizar a conversão dos dados de object para *datetime*. Na figura A.5 é possível verificar a função utilizada, assim como uma verificação posterior para confirmar a conversão.

**Figura A.5 – Conversão da coluna ‘Data’**

```
[ ] dfs = [df_alto_tiete, df_alto_tiete_res, df_cantareira, df_cantareira_res,\n        df_cotia, df_cotia_res, df_guarapiranga, df_guarapiranga_res, df_rio_claro,\n        df_rio_claro_res, df_rio_grande, df_rio_grande_res, df_sao_lourenco,\n        df_sao_lourenco_res]\n\n▶ for df in dfs:\n    df['Data'] = pd.to_datetime(df['Data'], dayfirst=True)\n    df.info()\n\n❸ <class 'pandas.core.frame.DataFrame'>\nRangeIndex: 8684 entries, 0 to 8683\nData columns (total 6 columns):\n #   Column           Non-Null Count  Dtype \n---  --\n 0   Data             8684 non-null    datetime64[ns]\n 1   Volume (hm³)     8684 non-null    float64\n 2   Volume (%)       8684 non-null    float64\n 3   Chuva (mm)      6857 non-null    float64\n 4   Vazão natural (m³/s) 8684 non-null    float64\n 5   Vazão a jusante (m³/s) 8684 non-null    float64\n dtypes: datetime64[ns](1), float64(5)\nmemory usage: 407.2 KB\n<class 'pandas.core.frame.DataFrame'>\nRangeIndex: 48048 entries, 0 to 48047\nData columns (total 8 columns):\n #   Column           Non-Null Count  Dtype \n---  --\n 0   Data             48048 non-null    datetime64[ns]\n 1   Nível            46061 non-null    float64\n 2   Volume (hm³)     39606 non-null    float64\n 3   Volume (%)       39606 non-null    float64\n 4   Chuva (mm)      41593 non-null    float64\n 5   Vazão natural (m³/s) 39606 non-null    float64\n 6   Vazão a jusante (m³/s) 39639 non-null    float64\n 7   Reservatório    48048 non-null    object\n dtypes: datetime64[ns](1), float64(6), object(1)\nmemory usage: 2.9+ MB
```

Fonte: Elaborada pelos autores (2023).

Outra etapa importante para que não haja inconsistências na análise dos dados é a verificação de valores nulos presentes dentro de um *Dataframe*. Para tal ação, é possível utilizar a função *isnull* mais uma função de soma para termos ciência do total contabilizado (figura A.6).

**Figura A.6** – Contagem dos valores nulos.

```
df_nulls.isnull().sum()
```

```
VrNivel      0  
VlVolumeHm   0  
VrVolume%    0  
VlQJusante   0  
VlQNatural   0  
VrChuva     0  
DdDia        0  
MmMes        0  
AaAno        0  
dtype: int64
```

Fonte: Elaborada pelos autores (2023).

Assim como valores nulos podem afetar a leitura dos dados, dados duplicados também são uma preocupação pertinente. Dessa forma, pode-se utilizar as funções *duplicate* e *drop\_duplicate* para a verificação de dados duplicados e sua eliminação respectivamente. Na figura A.7, temos a verificação de duplicidade dos nomes e valores das colunas, e na figura A.8 temos o desenvolvimento de uma função para nos retornar a quantidade de linhas duplicadas no *Dataframe* original e a quantidade de linhas após a verificação e remoção dos dados duplicados.

**Figura A.7** – Verificação de duplicidade nos nomes e valores das colunas

```
[ ] # Verificar se não há nomes de colunas duplicados
for df in dfs:
    colunas_nomes_iguais = (df.columns.duplicated() == False).all()
    if colunas_nomes_iguais:
        print(f'Não há nomes de colunas duplicados em {df.name}')
    else:
        print(f'Há nomes de colunas duplicados em {df.name}')

Não há nomes de colunas duplicados em df_alto_tiete
Não há nomes de colunas duplicados em df_alto_tiete_res
Não há nomes de colunas duplicados em df_cantareira
Não há nomes de colunas duplicados em df_cantareira_res
Não há nomes de colunas duplicados em df_cotia
Não há nomes de colunas duplicados em df_cotia_res
Não há nomes de colunas duplicados em df_guarapiranga
Não há nomes de colunas duplicados em df_guarapiranga_res
Não há nomes de colunas duplicados em df_rio_claro
Não há nomes de colunas duplicados em df_rio_claro_res
Não há nomes de colunas duplicados em df_rio_grande
Não há nomes de colunas duplicados em df_rio_grande_res
Não há nomes de colunas duplicados em df_sao_lourenco
Não há nomes de colunas duplicados em df_sao_lourenco_res

# Verificar coluna por coluna para verificar se há colunas com valores iguais
for df in dfs:
    colunas_iguais = False
    for i in range(len(df.columns)):
        for j in range(i + 1, len(df.columns)):
            if (df.iloc[:, i] == df.iloc[:, j]).all():
                colunas_iguais = True
                print(f'As colunas de índice {i} e {j} têm os mesmos valores em {df.name}')

    if not colunas_iguais:
        print(f'Não há colunas com os mesmos valores em {df.name}')


Não há colunas com os mesmos valores em df_alto_tiete
Não há colunas com os mesmos valores em df_alto_tiete_res
Não há colunas com os mesmos valores em df_cantareira
Não há colunas com os mesmos valores em df_cantareira_res
Não há colunas com os mesmos valores em df_cotia
Não há colunas com os mesmos valores em df_cotia_res
Não há colunas com os mesmos valores em df_guarapiranga
Não há colunas com os mesmos valores em df_guarapiranga_res
Não há colunas com os mesmos valores em df_rio_claro
Não há colunas com os mesmos valores em df_rio_claro_res
Não há colunas com os mesmos valores em df_rio_grande
Não há colunas com os mesmos valores em df_rio_grande_res
Não há colunas com os mesmos valores em df_sao_lourenco
Não há colunas com os mesmos valores em df_sao_lourenco_res
```

Fonte: Elaborada pelos autores (2023).

**Figura A.8 – Verificação e remoção de duplicidade.**

```
[ ] # Criar DataFrame com registros removidos
for df in dfs:
    df_sem_duplicadas = df.drop_duplicates()
    print(f'{df.name}: O DataFrame original tem tamanho {len(df)}. O DataFrame sem duplicadas tem tamanho {len(df_sem_duplicadas)}.')

df_alto_tiete: O DataFrame original tem tamanho 8684. O DataFrame sem duplicadas tem tamanho 8684.
df_alto_tiete_res: O DataFrame original tem tamanho 48048. O DataFrame sem duplicadas tem tamanho 48048.
df_cantareira: O DataFrame original tem tamanho 8684. O DataFrame sem duplicadas tem tamanho 8684.
df_cantareira_res: O DataFrame original tem tamanho 51993. O DataFrame sem duplicadas tem tamanho 51993.
df_cotia: O DataFrame original tem tamanho 8684. O DataFrame sem duplicadas tem tamanho 8684.
df_cotia_res: O DataFrame original tem tamanho 26020. O DataFrame sem duplicadas tem tamanho 26020.
df_guarapiranga: O DataFrame original tem tamanho 8684. O DataFrame sem duplicadas tem tamanho 8684.
df_guarapiranga_res: O DataFrame original tem tamanho 26007. O DataFrame sem duplicadas tem tamanho 26007.
df_rio_claro: O DataFrame original tem tamanho 8684. O DataFrame sem duplicadas tem tamanho 8684.
df_rio_claro_res: O DataFrame original tem tamanho 8684. O DataFrame sem duplicadas tem tamanho 8684.
df_rio_grande: O DataFrame original tem tamanho 8684. O DataFrame sem duplicadas tem tamanho 8684.
df_rio_grande_res: O DataFrame original tem tamanho 14042. O DataFrame sem duplicadas tem tamanho 14042.
df_sao_lourenco: O DataFrame original tem tamanho 8591. O DataFrame sem duplicadas tem tamanho 8591.
df_sao_lourenco_res: O DataFrame original tem tamanho 8684. O DataFrame sem duplicadas tem tamanho 8684.
```

Conclusão: não há registros idênticos.

Fonte: Elaborada pelos autores (2023).

Por fim, temos a verificação dos *outliers*. Na figura A.9, temos uma função que retorna os *outliers* superiores e inferiores.

**Figura A.9 – Verificação de *outliers***

```
[ ] # Função que retorna os outliers de uma Series baseado nos limites superior e
# inferior
def outliers(series):
    df_desc = series.describe()
    q1 = df_desc['25%']
    q3 = df_desc['75%']
    aiq = q3 - q1
    li = q1 - 1.5 * aiq
    ls = q3 + 1.5 * aiq

    filtro = (series < li) | (series > ls)
    return series[filtro]

# Verificar a quantidade de outliers em cada DataFrame
for df in dfs:
    qtd_outliers = len(outliers(df['Volume (hm³)']))
    pct_outliers = qtd_outliers/len(df['Volume (hm³)'])
    print(f'Quantidade de outliers em {df.name}: {qtd_outliers}. Isso representa {pct_outliers:.2f}% do total.')

Quantidade de outliers em df_alto_tiete: 77. Isso representa 0.01% do total.
Quantidade de outliers em df_alto_tiete_res: 5881. Isso representa 0.12% do total.
Quantidade de outliers em df_cantareira: 280. Isso representa 0.03% do total.
Quantidade de outliers em df_cantareira_res: 3020. Isso representa 0.06% do total.
Quantidade de outliers em df_cotia: 0. Isso representa 0.00% do total.
Quantidade de outliers em df_cotia_res: 0. Isso representa 0.00% do total.
Quantidade de outliers em df_guarapiranga: 0. Isso representa 0.00% do total.
Quantidade de outliers em df_guarapiranga_res: 1277. Isso representa 0.05% do total.
Quantidade de outliers em df_rio_claro: 0. Isso representa 0.00% do total.
Quantidade de outliers em df_rio_claro_res: 0. Isso representa 0.00% do total.
Quantidade de outliers em df_rio_grande: 47. Isso representa 0.01% do total.
Quantidade de outliers em df_rio_grande_res: 47. Isso representa 0.00% do total.
Quantidade de outliers em df_sao_lourenco: 140. Isso representa 0.02% do total.
Quantidade de outliers em df_sao_lourenco_res: 128. Isso representa 0.01% do total.
```

Fonte: Elaborada pelos autores (2023).