Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Estatística
Programa de Especialização em *Data Science* e *Big Data*

Matheus Allan Toledo

Abordagens e Estratégias Para Superar Desafios Na Exploração De Dados: Um Estudo De Caso

Curitiba
2023

Matheus Allan Toledo

Abordagens e Estratégias Para Superar Desafios Na Exploração De Dados: Um Estudo De Caso

Monografia apresentada ao Programa de Especialização em *Data Science* e *Big Data* da Universidade Federal do Paraná como requisito parcial para a obtenção do grau de especialista.

Orientador: Prof. André Ricardo Abed Grégio

Curitiba
2023

Abordagens e Estratégias Para Superar Desafios Na Exploração De Dados: Um Estudo De Caso

Approaches and Strategies to Overcome Challenges in Data Exploration

Matheus A. Toledo[1]

[1]Departamento de Estatística, Universidade Federal do Paraná R. Evaristo F. Ferreira da Costa 408, Jardim das Américas, 81530-0150, Curitiba, PR, Brasil*

A utilização de arquivos CSV (*Comma-Separated Values*) é uma prática comum em muitos campos da ciência devido à sua simplicidade e facilidade de manipulação. No entanto, a leitura de arquivos CSV pode ser um desafio significativo devido a uma série de problemas comuns que surgem durante o processo. Esta monografia analisa os desafios comuns enfrentados ao ler arquivos CSV e apresenta estratégias para solucioná-los. Os desafios incluem a diversidade de *encodings* utilizados nos arquivos, a leitura de arquivos grandes e a inconsistência de cabeçalhos e dados. Para resolver esses problemas, são discutidas técnicas como detecção e leitura correta com *encodings*, abordagens eficientes para a leitura de arquivos grandes e métodos para lidar com dados inconsistentes, como valores nulos, mudanças no nome de colunas e perda de colunas. O objetivo é fornecer aos leitores um conjunto de estratégias eficazes e eficientes para enfrentar esses desafios ao lidar com arquivos CSV.
Palavras-chave: CSV (*Comma-Separated Values*), *encodings* diferentes, arquivos grandes, análise geral, qualidade dos dados

The use of CSV (Comma-Separated Values) files is a common practice in many fields of science and industry, due to their simplicity and ease of manipulation. However, reading CSV files can be a significant challenge due to a series of common problems that arise during the process. This monograph examines the common challenges faced when reading CSV files and presents strategies to address them. These challenges include the diversity of *encodings* used in the files, reading very large files, and data inconsistency. To address these problems, techniques such as detecting and correctly converting *encodings*, efficient approaches for reading large files, and methods for handling inconsistent data such as null values, column name changes, and column loss are discussed. The aim is to provide readers with a set of effective and efficient strategies to tackle these challenges when dealing with CSV files.
Keywords: CSV (Comma-Separated Values), different *encodings*, large files, general analysis, data quality

1. Introdução

A pandemia de Covid-19 iniciada no final de 2019 trouxe uma série de desafios à saúde pública e à sociedade em geral. Diante dessa crise global de saúde, pesquisas e análises são essenciais para ajudar a entender os dados relacionados a doenças e auxiliar na tomada de decisões com base em evidências.

Nesse contexto, o objetivo principal desta monografia é abordar as etapas fundamentais da ciência de dados utilizando dados de *reports* de casos de Covid-19 no estado do Paraná. O estudo foca no entendimento do negócio, coleta de dados, pré-processamento e início das fases de análise exploratória dos dados.

Para obtenção dos dados necessários, foram utilizados os arquivos CSV disponibilizados pela Secretaria de Estado da Saúde do Paraná (SESA-PR)("Coronavírus COVID-19", s.d.), contendo registros de notificações de Covid-19 referentes ao período de 2020 a 2022. Esses arquivos foram escolhidos devido ao intervalo de tempo e relevância do assunto.

A coleta de dados é realizada por meio de *web scraping*, técnica que permite extrair informações de páginas da web de forma automatizada. Essa abordagem permite acesso eficiente e contínuo a dados atualiza-

*ttoledomatheus@gmail.com

dos, o que ajuda a construir conjuntos de dados robustos e representativos.

Após a coleta, os cabeçalhos do arquivo são analisados para identificar possíveis inconsistências. Portanto, o processamento é feito para ler os dados corretamente, considerando o cabeçalho mais comum como o correto. Esta etapa é fundamental para garantir a integridade e confiabilidade dos dados utilizados na análise.

Considerando o grande número de arquivos e a quantidade de informações neles contidas, optou-se pela utilização da biblioteca Dask(Dask Development Team, 2021) para leitura dos dados. O Dask é uma ferramenta que permite trabalhar com conjuntos de dados maiores que a memória disponível, realizando operações de forma paralela e distribuída.

Em seguida, foram identificados os dados que precisam ser processados com o objetivo de eliminar inconsistências, valores ausentes ou dados inválidos. Após o pré-processamento, obtém-se um *data frame* pronto para a etapa de análise exploratória.

A análise exploratória inicial concentra-se em duas colunas de dados principais. Esta etapa preliminar fornece uma compreensão inicial do conjunto de dados e orienta a próxima fase de análise.

Desta forma, esta monografia visa fornecer uma visão geral das etapas iniciais de compreensão operacional, coleta de dados, pré-processamento e análise exploratória do clico da ciência de dados. Por meio dessas fases, busca-se elaborar estratégias eficientes que possam ajudar em outros tipos de dados.

2. *Web Scraping* no Site da SESA/PR para Obtenção de Dados de Casos de COVID-19

O advento da pandemia de COVID-19 destacou a importância do monitoramento e análise dos casos diários da doença para a tomada de decisões em saúde pública. A obtenção automatizada de dados de fontes oficiais se tornou uma ferramenta valiosa nesse contexto. Neste capítulo, descreveremos o processo de *web scraping* realizado no site da SESA/PR para a extração de arquivos CSV contendo informações sobre casos diários de COVID-19. Utilizamos a biblioteca Selenium(“Selenium”, 2021) para automatizar as interações necessárias, incluindo a expansão de elementos *accordion* em HTML (Linguagem de Marcação de HiperTexto), que são componentes utilizados para organizar e exibir conteúdo de forma expansível e recolhível, permitindo a visualização dos *links*, e a renomeação dos arquivos para o formato dd-mm-yyyy.

2.1. Coleta de Dados

A coleta dos dados foi realizada por meio do *web scraping*, utilizando a biblioteca Selenium em conjunto com um navegador web automatizado. O Selenium nos permite simular interações humanas em uma página da *web*, possibilitando o acesso aos dados desejados.

Após analisar a estrutura do site da SESA/PR, identificamos que os *links* para os arquivos CSV estavam ocultos dentro de *accordions*. Portanto, utilizamos o Selenium para realizar cliques nos *accordions*, expandindo-os e tornando os *links* visíveis para posterior extração.

2.2. Expansão dos *Accordions*

A expansão dos *accordions* foi um passo crucial para obter acesso aos links para os arquivos CSV desejados. Utilizando as funcionalidades do Selenium, localizamos os elementos HTML correspondentes aos *accordions* e realizamos cliques programaticamente. Dessa forma, os *accordions* foram expandidos, revelando os *links* para os arquivos contendo os dados de casos diários de COVID-19.

2.3. Obtenção dos URLs dos Arquivos CSV

Após a expansão dos *accordions*, procedemos à extração dos URLs dos arquivos CSV. Utilizando as funcionalidades do Selenium, localizamos os elementos HTML que continham os *links* e extraímos os URLs correspondentes. Esses URLs foram armazenados para posterior download dos arquivos.

2.4. Renomeação dos Arquivos

Para melhorar a organização e facilitar a identificação dos arquivos, realizamos a renomeação dos arquivos para o formato dd-mm-yyyy, utilizando as datas correspondentes aos dados contidos em cada arquivo. Essa etapa foi importante para garantir a consistência dos dados coletados e facilitar sua utilização posterior.

2.5. Algoritmo

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import
    WebDriverWait
from selenium.webdriver.support import
    expected_conditions as EC
```

```
5  import re
6  import os
7  import requests
8
9  def extract_date_from_string(string):
10     # Regular expression pattern
11     pattern = r"_(\d{2}_\d{2})_"  # Matches "
       _dd_mm_"
12
13     match = re.search(pattern, string)
14     if match:
15         date_str = match.group(1).replace("_", "-
       ")  # Convert "_" to "-"
16         return date_str
17
18     return None  # Return None if no date is
       found
19
20
21 URL = 'https://www.saude.pr.gov.br/Pagina/Boletim
       -COVID19-'
22
23 for year in ['2020', '2021', '2022']:
24     # Set up the web driver
25     driver = webdriver.Chrome()
26     driver.get(URL + year)
27
28     # Wait for the page to load and for the first
        "spoiler-toggle" element to become visible
29     wait = WebDriverWait(driver, 10)
30     wait.until(EC.visibility_of_element_located((
       By.CLASS_NAME, 'spoiler-title')))
31
32     # Click on each "spoiler-toggle" element to
       reveal the CSV links
33     spoiler_toggles = driver.find_elements(By.
       CLASS_NAME, 'spoiler-title')
34     for toggle in spoiler_toggles:
35         try:
36             toggle.click()
37         except:
38             driver.execute_script("arguments[0].
       click();", toggle)
39
40     # Find the <a> tag with the "Geral" label and
        an href that ends with '.csv'
41     csv_links = driver.find_elements(By.XPATH, '
       //a[contains(text(),"Geral") and contains(
       @href,".csv")]')
42
43     # Create a directory to save the CSV files in
44     os.makedirs('/Users/matheustoledo/Documents/
       dsbd/monografia/csvs_sesa/geral/data/tmp/' +
       year, exist_ok=True)
45
46     # Download each CSV file and save it in the
       covid_csv directory
47     for link in csv_links:
48         file_url = link.get_attribute('href')
49         try:
50             file_name = f'{
       extract_date_from_string(file_url)}-{year}.
       csv'
51             with open(os.path.join(f'../data/tmp
       /{year}', file_name), 'wb') as file:
52                 response = requests.get(file_url)
53                 file.write(response.content)
54             print(file_name)
55         except:
56             print(file_url)
57
58     # Close the web driver
59     driver.quit()
```

2.5.1. Descrição do algoritmo

Este algoritmo é projetado para extrair dados do COVID-19 em formato CSV do site da Secretaria de Estado da Saúde do Paraná (SESA) para os anos de 2020, 2021 e 2022. Os dados são armazenados em elementos "spoiler-toggle"na página da web, que precisam ser clicados para revelar os links de download do CSV. O script usa a biblioteca Selenium para interagir com a página da Web e o Chrome WebDriver para automatizar o processo de clicar nas alternâncias e baixar os arquivos CSV.

2.5.2. Entendendo o algoritmo

1. Importação de bibliotecas (Linhas 1-7): O algoritmo importa as bibliotecas necessárias para o funcionamento do código. Isso inclui o Selenium, responsável pela automação do navegador, e outras bibliotecas para manipulação de elementos, expressões regulares, gerenciamento de arquivos e requisições HTTP.
2. Definição da URL (Linha 21): O algoritmo define a URL base do site de onde os boletins COVID-19 serão extraídos.
3. *Loop* pelos anos (Linhas 23-38): O algoritmo inicia um loop para iterar sobre uma lista de anos.
4. Configuração do *WebDriver* (Linha 25): O algoritmo configura o *WebDriver* do Selenium para utilizar o navegador Chrome. Essa configuração permite abrir a página *web* e interagir com seu conteúdo.
5. Aguardar o carregamento da página (Linhas 27-29): Utilizando o objeto *WebDriverWait*, o algoritmo espera até que um elemento com a classe *'spoiler-title'* se torne visível na página. Isso garante que a página tenha carregado completamente antes de prosseguir.

6. Clicar nos elementos *"spoiler-toggle"* (Linhas 32-37): O algoritmo localiza todos os elementos da página com a classe *'spoiler-title'* e clica em cada um deles para revelar os *links* para os arquivos CSV. Se ocorrer algum erro ao clicar em um elemento, o algoritmo utiliza a função *execute_script* para executar um clique usando JavaScript.
7. Localizar *links* para os arquivos CSV (Linha 40): O algoritmo utiliza XPath para localizar os elementos <a> que contêm o texto "Geral"e possuem um atributo href que termina com '.csv'. Esses *links* representam os arquivos CSV desejados.
8. Criar diretório para salvar os arquivos CSV (Linha 43): O algoritmo cria um diretório com o caminho '/csvs_sesa/geral/data/' seguido pelo ano atual. Esse diretório será utilizado para armazenar os arquivos CSV baixados.
9. *Download* dos arquivos CSV (Linhas 46-57): O algoritmo itera sobre os *links* para os arquivos CSV encontrados e realiza o download de cada um deles. Para cada *link*, é obtida a URL do arquivo CSV e, em seguida, é feito o *download* do arquivo utilizando a biblioteca *requests*. O arquivo é salvo no diretório criado anteriormente.
10. Fechar o *WebDriver* (Linha 60): Após concluir o *loop* pelos anos e baixar todos os arquivos CSV desejados, o algoritmo fecha o *WebDriver*, encerrando o navegador.

3. Descrição do *Dataset*

Neste capítulo, apresentaremos uma descrição detalhada do *dataset* utilizado, que consiste nos arquivos CSV obtidos por meio do *web scraping* descritos no capítulo anterior. Esses arquivos contêm informações sobre os casos diários de COVID-19, incluindo dados como data do diagnóstico, idade, sexo e outras variáveis relevantes. Exploraremos a estrutura dos arquivos, as variáveis contidas neles e discutiremos a relevância desses dados para análises e pesquisas relacionadas à pandemia.

3.1. Estrutura do *Dataset*

Cada arquivo CSV representa os casos diários de COVID-19 em uma determinada data. A estrutura dos arquivos é organizada em colunas, com cada coluna representando uma variável específica.

3.2. Relevância do *Dataset*

O *dataset* de casos diários de COVID-19 no Estado do Paraná obtido por meio do *web scraping* apresenta grande relevância para análises e pesquisas relacionadas à pandemia. Esses dados permitem um acompanhamento temporal da evolução da doença, identificação de tendências e variações nos números de casos confirmados, óbitos, recuperações e hospitalizações ao longo do tempo.

Além disso, esses dados podem ser utilizados para análises estatísticas, modelagem preditiva e estudos epidemiológicos. Pesquisadores e profissionais da saúde podem explorar o *dataset* para investigar fatores associados à propagação da COVID-19, identificar padrões geográficos e demográficos, avaliar a eficácia de medidas de controle e planejar estratégias de intervenção.

É importante ressaltar que o *dataset* possui informações atualizadas, uma vez que o *web scraping* foi realizado com o objetivo de obter os dados mais recentes disponibilizados pela SESA/PR.

4. Análise dos *headers* nos Arquivos CSV

Neste capítulo, apresentaremos uma análise detalhada dos *headers* presentes nos arquivos CSV. Ao explorar esses *headers*, descobrimos informações importantes sobre os *encodings* dos CSVs, bem como mudanças nos *headers* ao longo do tempo. Utilizaremos a biblioteca chardet para identificar os diferentes *encodings* e aplicamos técnicas de processamento de texto e análise de dados para extrair *insights* relevantes. Além disso, geramos gráficos para visualizar as mudanças nos *headers* e identificar padrões entre as colunas.

4.1. Identificação dos *encodings* dos CSVs

Para identificar os diferentes *encodings* presentes nos CSVs, utilizamos a biblioteca chardet. Essa biblioteca analisa os *bytes* dos arquivos e estima o encoding mais provável. Para evitar uma enorme quantidade de dados na memória RAM lemos apenas as 10.000 primeiras linhas de cada CSV.

Após essa análise verificamos que existe um total de 4 diferentes *encodings* e então criamos um dicionário que associa cada arquivo CSV ao seu respectivo encoding, permitindo uma leitura correta dos dados.

4.1.1. Algoritmo

Tabela 1: *encodings* identificados

encodings
Windows-1252
utf-8
ISO-8859-1
UTF-8-SIG

```
1  import chardet
2
3  def detect_encoding(file):
4    with open(file, 'rb') as raw_data:
5      result = chardet.detect(raw_data.read(10000))
6      return result['encoding']
7
8  def get_file_encoding(file_path):
9      file_encoding = detect_encoding(file_path)
10     return file_encoding
```

Listing 1: Detecção De *encodings*

```
1  FILE_\textit{encodings} = {
2      '2020': [],
3      '2021': [],
4      '2022': []
5  }
6  for year in FILE_PATHS:
7    \textit{encodings} = {}
8    for path in FILE_PATHS[year]:
9      encoding = get_file_encoding(path)
10     file_name = path.split('/')[-1]
11     \textit{encodings}[path] = encoding
12   FILE_\textit{encodings}[year] = \textit{
       encodings}
```

Listing 2: Dicionário De *encodings*

4.1.2. Descriçao do algoritmo Detecção De *encodings*

Este algoritmo em Python utiliza a biblioteca chardet para detectar a codificação de caracteres de um arquivo. A codificação de caracteres refere-se à maneira como os caracteres são representados internamente em um arquivo de texto.

4.1.3. Entendendo o algoritmo Detecção De *encodings*

1. Function *detect_encoding* (Linhas 1-6): Essa função recebe um arquivo como entrada e detecta a codificação do conteúdo do arquivo. Ela abre o arquivo em modo binário usando a função open e lê os primeiros 10.000 *bytes* de dados brutos. Em seguida, a biblioteca chardet é utilizada para analisar os dados brutos e determinar a codificação. A codificação detectada é retornada como resultado.
2. Função *get_file_encoding* (Linhas 8-10): Essa função recebe o caminho de um arquivo como entrada e chama a função *detect_encoding* para determinar a codificação do arquivo. A codificação detectada é armazenada na variável *file_encoding* e é retornada como resultado da função.

4.1.4. Descriçao do algoritmo Dicionário De *encodings*

Este algoritmo cria um dicionário chamado *FILE_encodings* que tem três chaves: '2020', '2021' e '2022'. Cada chave está associada a uma lista de *encodings* para os arquivos CSV de cada respectivo ano.

4.1.5. Entendendo o algoritmo Dicionário De *encodings*

1. Linha 1: Define um dicionário vazio chamado *FILE_encodings*.
2. Linhas 2-4: São adicionadas três chaves ao dicionário *FILE_encodings*: '2020', '2021' e '2022', cada uma associada a uma lista vazia.
3. Linha 6: Inicia-se um loop for que itera sobre as chaves do dicionário *FILE_PATHS*.
4. Linha 7: É criado um dicionário vazio chamado *encodings* para armazenar os pares de caminho de arquivo e codificação.
5. Linha 8: Inicia-se um *loop* for que itera sobre os caminhos de arquivo associados ao ano atual.
6. Linha 9: Chama-se a função *get_file_encoding* para obter a codificação do arquivo.
7. Linha 10: Extrai-se o nome do arquivo do caminho.
8. Linha 11: Adiciona-se uma entrada ao dicionário *encodings*, mapeando o caminho do arquivo para sua codificação.
9. Linha 12: Atualiza-se a lista de codificações para o ano atual no dicionário *FILE_encodings*, associando a chave correspondente ao dicionário *encodings*.
10. Linhas 6-12: O *loop* continua, iterando sobre as chaves restantes em *FILE_PATHS*, repetindo o processo para cada ano.

4.2. Criação do *DataFrame* para Análise dos *headers*

Com os *encodings* identificados, criamos um *DataFrame* contendo as colunas ["Path", "Header anterior", "Novo header", "Mês", "Ano"]. O DataFrame contém informação referente às mudanças de header de um arquivo para outro.

Essa estrutura nos permitirá comparar os *headers* anteriores com os *headers* atuais e analisar as mudanças ocorridas.

Tabela 2: Colunas do dataframe utilizado na análise dos *headers*

Colunas	Tipo
Path	String
Arquivo	String
Header anterior	String
Novo header	String
Mês	String
Ano	String

4.3. Análise das Mudanças de *headers* por Ano

Utilizando o *DataFrame* criado, realizamos uma análise para quantificar as mudanças nos *headers* ao longo do tempo. Descobrimos que houve um total de 282 mudanças de *headers* nos arquivos de 2020, 2021 e 2022. Mais especificamente, identificamos que o ano de 2020 apresentou 34 mudanças, o ano de 2021 teve 149 mudanças e o ano de 2022 registrou 99 mudanças.

Para visualizar as mudanças de *headers* de forma mais clara, geramos um gráfico que exibe a distribuição das mudanças ao longo dos anos. Esse gráfico nos permite identificar períodos em que ocorreram mudanças significativas nos *headers* e facilita a compreensão das tendências.

4.4. *headers* Mais Frequentes

Além de analisar as mudanças de *headers*, também examinamos quais *headers* aparecem com mais frequência nos arquivos. Isso nos ajuda a identificar padrões e entender quais informações são consistentes ao longo do tempo.

4.5. Análise de Similaridade de Nomes de Colunas

Para identificar colunas com nomes parecidos que podem indicar mudanças de nomes, realizamos uma análise comparativa utilizando cálculo de distância

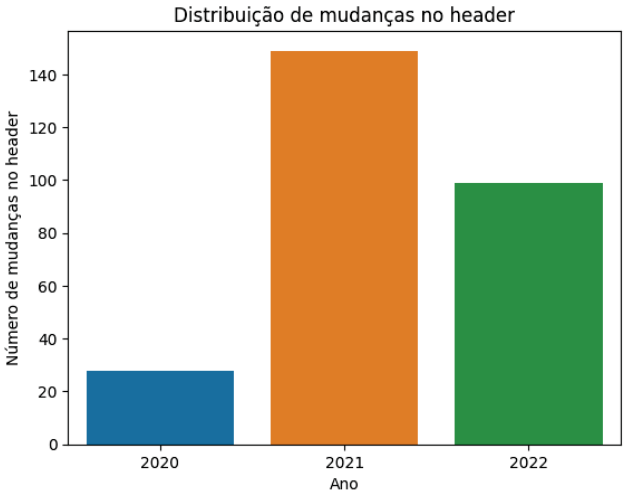

Figura 1: Número de mudanças de *headers* por ano

Tabela 3: Colunas removidas e adicionadas nos *headers*

Colunas	Situação
DATA_OBITO_DIVULGACAO	Adicionada
MUN_ATENDIMENTO	Adicionada
MUN_RESIDENCIA	Adicionada
LABORATORIO	Adicionada
DATA_CONFIRMACAO_DIVULGACAO	Adicionada
FONTE_DADO_RECUPERADO	Adicionada
DT_ATUALIZACAO	Removida
Mun_atend	Removida
dt_com_obito	Removida
Laboratorio	Removida
dt_notificacao	Removida
Mun_Resid	Removida
Fonte	Removida

de Levenshtein. Essa técnica nos permite identificar a ortografia de colunas que são semelhantes, mas não idênticas. Além disso, utilizamos a biblioteca wordnet para analisar a semântica das colunas, identificando conjuntos de sinônimos relacionados. Com base nessas informações, geramos um gráfico de *network* que exibe as colunas com nomes parecidos, facilitando a identificação de padrões e mudanças nas nomenclaturas.

4.6. Conclusão da análise dos *headers*

A análise detalhada dos *headers* nos arquivos CSV proporcionou insights valiosos sobre os dados e sua evolução ao longo do tempo. Ao identificar os diferentes *encodings* dos CSVs, garantimos a leitura correta

Tabela 4: Header mais frequente nos arquivos CSV

Colunas
IBGE_RES_PR
IBGE_ATEND_PR
UF_RESIDENCIA
SEXO
IDADE_ORIGINAL
MUN_RESIDENCIA
MUN_ATENDIMENTO
LABORATORIO
DATA_DIAGNOSTICO
DATA_CONFIRMACAO_DIVULGACAO
DATA_INICIO_SINTOMAS
OBITO
DATA_OBITO
DATA_OBITO_DIVULGACAO
STATUS
DATA_RECUPERADO_DIVULGACAO
ORIGEM_NOTIFICACAO

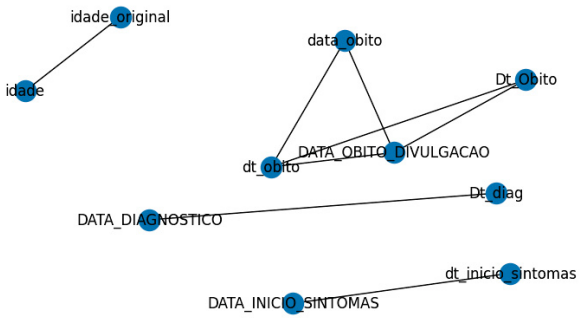

Figura 2: Similaridade de nomes de colunas

dos dados. A análise das mudanças de *headers* por ano revelou padrões e períodos de alterações mais significativas. Os gráficos gerados permitiram visualizar de forma clara as mudanças e identificar os *headers* mais frequentes.

A análise de similaridade de nomes de colunas utilizando o cálculo de distância de Levenshtein e a análise semântica com a biblioteca wordnet nos ajudou a identificar colunas com nomes parecidos, indicando possíveis mudanças de nomenclatura. O gráfico de *network* nos auxiliou na visualização dessas relações, destacando as colunas com nomes similares.

Essa análise dos *headers* dos arquivos CSVs contribui para uma melhor compreensão dos dados, permitindo a identificação de padrões, tendências e mudanças significativas. Essas informações são essenciais para uma análise de dados mais precisa e informada, facilitando a tomada de decisões com base nos insights obtidos.

5. Leitura dos dados

5.1. Lidando com Múltiplos Arquivos CSVs Utilizando a Técnica de *Lazy Loading* com Dask e Pandas

Lidar com grandes volumes de dados é um desafio comum em muitas aplicações de análise de dados. Quando os dados estão armazenados em vários arquivos CSV e o tamanho total desses arquivos excede a capacidade da memória RAM disponível, é necessário encontrar uma solução eficiente para realizar a análise dos dados. Neste capítulo, discutiremos o problema de lidar com múltiplos arquivos CSVs que totalizam mais de 200GB e como a técnica de *lazy loading*, utilizando as bibliotecas Dask e Pandas(McKinney, Wes, 2020), pode ser empregada como uma solução viável.

Arquivos CSV são uma forma popular de armazenar dados tabulares, pois são fáceis de serem lidos e manipulados. No entanto, quando temos um grande número de arquivos CSV, cada um contendo uma quantidade significativa de dados, a tarefa de ler todos esses arquivos para a memória RAM pode se tornar inviável. A limitação da memória RAM é um obstáculo que impede a análise direta dos dados.

5.2. A Técnica de *Lazy Loading* com Dask e Pandas

Uma solução para esse problema é a utilização da técnica de *lazy loading*, juntamente com as bibliotecas Dask e Pandas. O Dask é uma biblioteca Python projetada para lidar com grandes conjuntos de dados, permitindo a computação paralela e distribuída. O Pandas, por sua vez, é uma biblioteca amplamente utilizada para análise de dados em Python.

Passos para Implementar o *Lazy Loading* com Dask e Pandas: A implementação da técnica de *lazy loading* com Dask e Pandas para lidar com múltiplos arquivos CSVs de grande tamanho envolve os seguintes passos:

1. Uso eficiente da memória: O Dask é projetado para trabalhar com conjuntos de dados maiores do que a capacidade da memória RAM disponível. Ele realiza a leitura dos dados em blocos menores, permitindo processar apenas as partes necessárias para cada etapa de análise. Isso evita sobrecarregar a memória RAM e torna possível trabalhar com grandes conjuntos de dados mesmo em ambientes com recursos limitados.

2. Computação paralela e distribuída: O Dask oferece suporte à computação paralela e distribuída, o que permite acelerar o processamento de grandes volumes de dados. Ele pode aproveitar recursos como CPUs *multi-core*, *clusters* de computação e até mesmo serviços em nuvem, distribuindo as tarefas de processamento entre várias unidades de processamento de forma eficiente.
3. Integração com o Pandas: O Pandas é amplamente utilizado para análise de dados em Python devido à sua rica funcionalidade e facilidade de uso. Ao utilizar o Dask em conjunto com o Pandas, é possível aproveitar todas as operações e transformações disponíveis no Pandas, aplicando-as aos blocos de dados carregados pelo Dask. Isso proporciona uma experiência familiar aos usuários do Pandas e simplifica o processo de análise.
4. Escalabilidade: A combinação do Dask e do Pandas oferece uma solução escalável para lidar com grandes conjuntos de dados. À medida que o tamanho dos arquivos ou o número de arquivos aumenta, é possível dimensionar o processamento utilizando recursos adicionais, como mais CPUs ou nós de computação em um *cluster*. O Dask gerencia automaticamente a divisão e distribuição dos dados, permitindo que a análise seja realizada de maneira eficiente.

5.3. Problemas Encontrados durante a Leitura dos Arquivos CSV

5.4. Identificação do *Header* Correto

Durante a leitura dos arquivos CSV, identificamos que os cabeçalhos nem sempre estavam consistentes entre os diferentes arquivos. Algumas variações no nome das colunas foram encontradas, o que dificultou a identificação do cabeçalho correto a ser considerado para a estruturação do *dataset*.

5.5. Desorganização das Colunas no *Dataset*

Outro problema encontrado foi a desorganização das colunas no *dataset* resultante da leitura dos arquivos CSV. O Pandas, por padrão, considera a primeira linha do arquivo como o número correto de colunas. No entanto, devido às variações nos cabeçalhos mencionadas anteriormente, isso resultou em um dataset com colunas desalinhadas e dados inconsistentes.

5.6. Solução Adotada

Para solucionar esses problemas, adotamos a seguinte abordagem:

1. Leitura dos Arquivos CSV com *Header* = *None*: Inicialmente, realizamos a leitura dos arquivos CSV utilizando o parâmetro *"header = None"* do Pandas. Isso permite que o Pandas não assuma automaticamente a primeira linha como cabeçalho correto, evitando problemas de alinhamento de colunas.
2. Identificação do *Header* Mais Frequente: Após a leitura dos arquivos sem o cabeçalho definido, exploramos os dados para identificar o cabeçalho mais frequente presente nos arquivos. Utilizamos técnicas estatísticas e contagem de ocorrências para determinar o cabeçalho mais comum entre os arquivos.
3. Atribuição das Colunas ao *DataFrame*: Com base no cabeçalho mais frequente identificado, atribuímos corretamente as colunas ao *DataFrame*. Utilizamos o método "df.columns"do Pandas para definir o cabeçalho correto, garantindo a consistência e organização adequadas das colunas.

5.7. Algoritmo

```
import numpy as np
import pandas as pd
import csv
import io

def read_data(file_path, year):
  # Use the most frequent header
  dtypes = {
      'IBGE_RES_PR': str,
      'IBGE_ATEND_PR': str,
      'UF_RESIDENCIA': str,
      'SEXO': str,
      'IDADE_ORIGINAL': str,
      'MUN_RESIDENCIA': str,
      'MUN_ATENDIMENTO': str,
      'LABORATORIO': str,
      'DATA_DIAGNOSTICO': str,
      'DATA_CONFIRMACAO_DIVULGACAO': str,
      'DATA_INICIO_SINTOMAS': str,
      'OBITO': str,
      'DATA_OBITO': str,
      'DATA_OBITO_DIVULGACAO': str,
      'STATUS': str,
      'DATA_RECUPERADO_DIVULGACAO': str,
      'ORIGEM_NOTIFICACAO': str
  }

```

```
28  # Read only the CSV header
29  try:
30      with open(file_path, "r") as file:
31          reader = csv.reader(file)
32          header = next(reader)[0].split(';')
33  except:
34      print(file_path)
35
36  # Get the encoding from the dictionary
37  detected_encoding = FILE_\textit{encodings}[
      year][file_path]
38
39  try:
40      # Try to read the file with pandas *Note
      the header=None to avoid problems with
      misaligned data
41      df = pd.read_csv(file_path, header=None,
      engine='python', on_bad_lines='skip',
      delimiter=';', encoding=detected_encoding)
42      # Assign the columns to the dataframe
43      df.columns = header
44  except:
45      # If it fails, create an empty dataframe
      with the considered good header's dtypes
46      df = pd.read_csv(io.StringIO(''), names=
      dtypes.keys(), dtype=dtypes)
47
48  for column in df.columns:
49      # Drop the columns that we will not
      consider
50      if column not in dtypes.keys():
51          df.drop(column, axis=1, inplace=True)
52
53  for column in dtypes.keys():
54      # Assign NaN to the missing columns
55      if column not in df.columns:
56          df[column] = np.NaN
57
58  df = df.iloc[1:]
59
60  # Order of the columns
61  names = ['IBGE_RES_PR', 'IBGE_ATEND_PR', '
      UF_RESIDENCIA', 'SEXO', 'IDADE_ORIGINAL', '
      MUN_RESIDENCIA', 'MUN_ATENDIMENTO', '
      LABORATORIO', 'DATA_DIAGNOSTICO', '
      DATA_CONFIRMACAO_DIVULGACAO', '
      DATA_INICIO_SINTOMAS', 'OBITO', 'DATA_OBITO',
       'DATA_OBITO_DIVULGACAO', 'STATUS', '
      DATA_RECUPERADO_DIVULGACAO', '
      ORIGEM_NOTIFICACAO']
62
63  # Assign data types, considering only strings
      for now
64  df = df[names]
65  df.IBGE_RES_PR = df.IBGE_RES_PR.astype(str)
66  df.IBGE_ATEND_PR = df.IBGE_ATEND_PR.astype(str)
67  df.UF_RESIDENCIA = df.UF_RESIDENCIA.astype(str)
68  df.SEXO = df.SEXO.astype(str)
69  df.IDADE_ORIGINAL = df.IDADE_ORIGINAL.astype(
      str)
70  df.MUN_RESIDENCIA = df.MUN_RESIDENCIA.astype(
      str)
71  df.MUN_ATENDIMENTO = df.MUN_ATENDIMENTO.astype(
      str)
72  df.LABORATORIO = df.LABORATORIO.astype(str)
73  df.DATA_DIAGNOSTICO = df.DATA_DIAGNOSTICO.
      astype(str)
74  df.DATA_CONFIRMACAO_DIVULGACAO = df.
      DATA_CONFIRMACAO_DIVULGACAO.astype(str)
75  df.DATA_INICIO_SINTOMAS = df.
      DATA_INICIO_SINTOMAS.astype(str)
76  df.OBITO = df.OBITO.astype(str)
77  df.DATA_OBITO = df.DATA_OBITO.astype(str)
78  df.DATA_OBITO_DIVULGACAO = df.
      DATA_OBITO_DIVULGACAO.astype(str)
79  df.STATUS = df.STATUS.astype(str)
80  df.DATA_RECUPERADO_DIVULGACAO = df.
      DATA_RECUPERADO_DIVULGACAO.astype(str)
81  df.ORIGEM_NOTIFICACAO = df.ORIGEM_NOTIFICACAO.
      astype(str)
82
83  return df
```

Listing 3: Leitura dos arquivos CSV

5.7.1. Descrição do algoritmo

O algoritmo é uma função chamada *read_data* que tem como objetivo ler um arquivo CSV, processá-lo e retornar um *DataFrame* do pandas com os dados processados.

5.7.2. Entendendo o algoritmo

1. Linhas 1-4: Importação das bibliotecas necessárias: numpy(Harris et al., 2020), pandas, csv e io.
2. Linhas 29-34: Tentativa de leitura do *header* do arquivo CSV especificado pelo *file_path*, separando-o por ponto e vírgula para obter os nomes das colunas.
3. Linha 37: Atribuição do *encoding* (codificação) do arquivo com base no dicionário *FILE_encodings* criado anteriormente.
4. Linhas 39-46: Tentativa de ler o arquivo CSV utilizando o pandas, especificando o separador como ponto e vírgula e o *encoding* detectado anteriormente. Se houver problemas de leitura de linhas mal formatadas, elas são ignoradas. Caso a leitura falhe (exceção), cria-se um *DataFrame* vazio com as colunas e tipos de dados definidos no dicionário dtypes.
5. Linhas 48-51: Realiza um *loop* nas colunas do *DataFrame* e remove aquelas que não estão presentes no dicionário dtypes, mantendo apenas as colunas de interesse.

6. Linhas 53-56: Realiza outro *loop* nas chaves do dicionário dtypes e atribui NaN (valor nulo) às colunas ausentes no *DataFrame*.
7. Linha 58: Remove a primeira linha do *DataFrame*, que é o cabeçalho do CSV lido.
8. Linha 61: Define a ordem das colunas do *DataFrame* através da lista names.
9. Linhas 64-81: Converte cada coluna do *DataFrame* para o tipo de dado especificado no dicionário dtypes.
10. Linha 83: Retorna o *DataFrame* processado.

5.8. Correção de Dados Importantes

A integridade dos dados é fundamental em qualquer pesquisa ou análise, e garantir a precisão e consistência de variáveis importantes é essencial para obter resultados confiáveis. Este capítulo foca na correção de duas variáveis críticas, nomeadas 'SEXO' e 'IDADE_ORIGINAL', no conjunto de dados em estudo. O objetivo é corrigir inconsistências e transformar os dados em um formato consistente e utilizável, contribuindo para melhorar a qualidade das análises subsequentes.

5.8.1. Correção dos Dados da Coluna 'SEXO'

A coluna 'SEXO' representa informações de gênero no conjunto de dados. Ao examinar os dados, foram identificados registros com valores incorretos ou não padronizados. Para abordar esse problema, foi realizado um processo de limpeza dos dados, resultando na correção da coluna 'SEXO' para considerar apenas entradas válidas para gênero, ou seja, 'M' (Masculino) e 'F' (Feminino).

5.8.2. Tratamento dos Dados da Coluna 'IDADE_ORIGINAL'

A coluna 'IDADE_ORIGINAL' contém dados relacionados à idade das pessoas no conjunto de dados. No entanto, foram observadas inconsistências na forma de representação, como *strings* contendo informações de idade, como '20 anos', ao invés do valor numérico esperado. Neste capítulo, é abordada a metodologia utilizada para converter esses formatos não padronizados em valores numéricos consistentes, tornando a coluna 'IDADE_ORIGINAL' em um tipo de dado homogêneo e utilizável. A relevância desse processo de transformação dos dados é discutida no contexto de garantir a consistência dos dados e facilitar as análises estatísticas subsequentes.

5.8.3. Algoritmo

```
1  def fix_idade_original(idade_original):
2    idade_original = idade_original.lower()
3    try:
4      if 'dias' in idade_original or 'dia' in
       idade_original:
5        return np.int8(float(idade_original.split()
       [0]) / 365.25)
6      elif 'meses' in idade_original or 'm s' in
       idade_original:
7        return np.int8(float(idade_original.split()
       [0]) / 12)
8      elif 'anos' in idade_original or 'ano' in
       idade_original:
9        return np.int8(float(idade_original.split()
       [0]))
10     else:
11       return np.int8(float(idade_original))
12   except:
13       return None
14
15 def simple_map(df):
16   df.IDADE_ORIGINAL = df.IDADE_ORIGINAL.map(
       fix_idade_original)
17   return df
```

Listing 4: Função para corrigir dados da coluna 'IDADE_ORIGINAL'

5.9. Conclusão Da Leitura Dos Dados

O problema de lidar com múltiplos arquivos CSVs de grande tamanho pode ser resolvido de forma eficiente utilizando a técnica de lazy loading com Dask e Pandas. Essa abordagem permite processar os dados sob demanda, carregando apenas partes dos arquivos para a memória RAM, economizando recursos e permitindo a análise de grandes volumes de dados. A combinação do Dask com o Pandas oferece uma solução escalável e flexível, permitindo realizar análises complexas em grandes conjuntos de dados de forma eficiente.

6. Lições Aprendidas na Leitura e Pré-processamento dos Dados

6.1. Entendimento da Importância da Qualidade dos Dados

A leitura e pré-processamento dos dados proporcionaram uma compreensão aprofundada sobre a relevância da qualidade dos dados para o sucesso da pesquisa. Identificar e corrigir inconsistências, dados

```
df
[39] ✓ 0.0s                                                                                    Python
```

	IBGE_RES_PR	IBGE_ATEND_PR	UF_RESIDENCIA	SEXO	IDADE_ORIGINAL	MUN_RESIDENCIA	MUN_ATENDIMENTO	LABORATORIO	DATA_DIAGNOSTICO	DATA_CONFIRMACAO_DIVULGACAO	DA
1	nan	4119103	nan	F	NaN	nan	nan	nan	15/07/2020	nan	
2	nan	4119905	nan	F	NaN	nan	nan	nan	18/06/2020	nan	
3	nan	4109807	nan	F	NaN	nan	nan	nan	27/07/2020	nan	
4	nan	4116901	nan	M	NaN	nan	nan	nan	27/07/2020	nan	
5	nan	4115804	nan	F	NaN	nan	nan	nan	15/07/2020	nan	
...	...	...	...	...	...	...	...	...	...	...	
131902	nan	4101804	nan	M	7.0	ARAUCARIA	ARAUCARIA	IBMP	30/08/2020	31/08/2020	
131903	nan	4104907	nan	F	58.0	CASTRO	CASTRO	IBMP	30/08/2020	31/08/2020	
131904	nan	4115200	nan	F	52.0	MARINGA	MARINGA	IBMP	31/08/2020	31/08/2020	
131905	nan	4126272	nan	F	56.0	SAUDADE DO IGUACU	SAUDADE DO IGUACU	IBMP	30/08/2020	31/08/2020	
131906	nan	4108403	nan	F	59.0	FRANCISCO BELTRAO	FRANCISCO BELTRAO	IBMP	30/08/2020	31/08/2020	

31566255 rows × 17 columns

Figura 3: *Dataframe* final após a leitura e pré-processamento

faltantes e formatos não padronizados são passos cruciais para garantir a confiabilidade e validade dos resultados obtidos.

6.2. Necessidade de Cuidados na Interpretação do *Header*

O processo de leitura do *header* dos arquivos CSV destacou a importância de interpretar corretamente as informações contidas nas colunas. A escolha do *header* mais frequente e a correspondência correta entre as colunas e seus respectivos tipos de dados foram fundamentais para a manipulação adequada dos dados.

6.3. Importância da Limpeza e Transformação dos Dados

A necessidade de limpeza e transformação dos dados ficou evidente durante o pré-processamento. Corrigir valores inválidos ou não padronizados em variáveis-chave, como 'SEXO' e 'IDADE_ORIGINAL', contribuiu para a consistência dos dados e facilitou as análises subsequentes.

6.4. Validação e Verificação de Dados

Durante o processo de pré-processamento, foi essencial verificar e validar os dados após a correção. A verificação de dados garantiu a eficácia das etapas anteriores e permitiu identificar possíveis erros ou omissões que poderiam impactar a integridade do conjunto de dados.

6.5. Impacto na Qualidade dos Resultados

As lições aprendidas na leitura e pré-processamento dos dados tiveram um impacto direto na qualidade dos resultados finais da pesquisa. A atenção aos detalhes, correção de erros e transformação adequada dos dados contribuíram para resultados mais confiáveis e significativos.

6.6. Considerações para Pesquisas Futuras

As experiências adquiridas durante a leitura e pré-processamento dos dados oferecem insights valiosos para pesquisas futuras. A importância de investir tempo e recursos na etapa inicial de preparação dos dados é essencial para alcançar resultados de alta qualidade e evitar problemas potenciais ao longo da análise.

7. Conclusão

A condução da pesquisa, desde a leitura até o pré-processamento dos dados, revelou a importância crucial de adotar uma abordagem minuciosa e metodológica para assegurar a qualidade dos dados utilizados. As valiosas lições aprendidas enfatizam a necessidade de corrigir inconsistências, tratar valores atípicos e transformar formatos não padronizados, resultando em um conjunto de dados coeso e confiável. A aplicação cuidadosa dessas lições proporcionou uma base sólida para as análises estatísticas e conclusões robustas, elevando significativamente o impacto e a credibilidade do estudo como um todo. O compromisso com a excelência na preparação dos dados estabeleceu uma sólida fundação para futuras pesquisas e reforçou a relevância da qualidade dos dados como pilar fundamental na obtenção de resultados científicos precisos e relevantes. Assim, ao valorizar a etapa inicial de tratamento dos dados, este estudo destaca a importância de contribuir para a excelência da pesquisa acadêmica

e sua significativa contribuição para o avanço do conhecimento em sua área de estudo.

Agradecimentos

Ao professor André Gregio, por ter sido meu orientador e ter desempenhado tal função com dedicação e amizade.
Aos meus pais e irmã, que me incentivaram nos momentos difíceis e compreenderam a minha ausência enquanto eu me dedicava à realização deste trabalho.
À minha amada namorada por todo o apoio e compreensão que me proporcionou ao longo do desenvolvimento desta monografia.
Ao meu estimado amigo Cláudio Torres Junior pelo seu inestimável apoio e orientação ao longo de todo o processo de programação.

Referências

Coronavírus COVID-19. (s.d.). Governo do Paraná. Recuperado 4 julho 2023, de https://www.saude.pr.gov.br/Pagina/Coronavirus-COVID-19

Dask Development Team. (2021). Dask: Parallel computing with task scheduling.

Harris, C. R. et al. (2020). NumPy: A fundamental package for scientific computing with Python. https://numpy.org/

McKinney, Wes. (2020). pandas: Powerful data analysis toolkit.

Selenium. (2021). *SeleniumHQ*. https://www.selenium.dev/