Análise de dados do covid-19 no Brasil usando métodos de ordenação

Murilo Holtz Foltran
Universidade Federal de São Paulo
São José dos Campos
murilo.holtz@unifesp.br

***Resumo* — Neste trabalho, foram utilizados dados disponibilizados pelo site Open Data, do Sistema Único de Saúde do Brasil, para observar e analisar padrões dentre o perfil de pacientes com casos leves a moderados suspeitos da Covid-19. Os dados foram implementados para a vigilância da Síndrome Gripal (SG) e para controle de casos possíveis do novo vírus. Neste trabalho, a base de dados utilizada é relacionada ao estado de São Paulo e contém 20 mil entradas de pacientes de diferentes regiões e idades.**

***Palavras-chave* — Análise de Dados, Covid-19, Algoritmos e Estrutura de Dados**

INTRODUÇÃO

Devido à pandemia do Coronavírus, a vigilância da Síndrome Gripal (SG) se tornou de grande importância para a cautela de casos do Covid-19. Com isso, o Ministério da Saúde, por meio da Secretaria de Vigilância em Saúde, implementou um banco de dados epidemiológicos que foram utilizados neste trabalho a fim de analisar e descobrir padrões dentre esses pacientes. Devido ao grande número de casos notificados da Covid-19, alguns estados possuem mais de um milhão de registros [1]. Assim, neste trabalho, a base de dados foi reduzida aleatoriamente para vinte (20) mil registros no estado de São Paulo entre março e outubro de 2020.

OBJETIVO

Neste trabalho, tem-se como objetivo estruturar análises de dados relacionados ao Covid-19, utilizando conceitos da computação, incluindo métodos de ordenação e a impressão dos dados a partir de imagens e gráficos. Além disso, tem-se também como objetivo utilizar as bibliotecas da linguagem escolhida para visualizar tais dados e informações importantes para o caso.

MATERIAIS E MÉTODOS

A implementação do código foi feita utilizando a linguagem de programação Python 3.8. Com ela, foram importadas algumas bibliotecas específicas da linguagem para auxílio em tabulações de dados e plotagem de gráficos. Além disso, foram implementados conceitos de listas, dicionários, definições de funções, quadros e estruturas de dados, gráficos e histogramas. O ambiente de desenvolvimento escolhido foi o Google Colab, caderno *online* de programação que não requer configurações externas e é executado na nuvem.

A máquina utilizada para a implementação dos códigos foi um hardware com processador Intel Core i5, memória instalada de 8gb e um Sistema Operacional instalado do Windows 10 de 64 bits.

A. Importações

A biblioteca padrão do Python é muito diversificada, oferecendo uma variedade enorme de recursos. Neste trabalho, foram utilizadas algumas ferramentas da biblioteca disponibilizada para implementação do código. Dentre elas, algumas foram importadas para possibilitar cálculos, plotagens, visualizações, etc.

Biblioteca: sys, math, random, matplotlib, cycler, numpy, pandas, auth, gspread, time e Counter.

B. Métodos

No início do caderno, foram definidas algumas funções de ordenação que serão utilizadas durante o desenvolvimento do trabalho. Dentre elas, as funções insertionSort( ), countingSort( ) e mergeSort( ). Logo após, foi aberto o *spreadsheet* com os dados que serão utilizados no trabalho. Com isso, foi utilizado uma biblioteca que conecta o Google Colab com o Google Sheets. Assim, após uma simples autenticação de usuário, é possível acessar os dados abertos no Google Sheets pelo caderno do Google Colab. Com o arquivo 'dados-sp-covid' aberto, foi inicializado o *worksheet* com todas as informações contidas no arquivo.

Com a biblioteca pandas, é atribuída a função DataFrame para visualizar os dados tabelados. O primeiro *array* construído, para análise e ordenação de dados, foi armazenando as idades dos pacientes registrados. Depois disso, foram adquiridos os dados de estado final dos pacientes, e relacionados com as idades de cada paciente. Para finalizar, foram filtrados pacientes curados e relacionados com suas respectivas idades. Para isso, foram realizadas algumas operações, como a transformação dos elementos do *array* de idade para números inteiros e a ordenação desses elementos com a função insertionSort( ), countingSort( ) e mergeSort( ). Para cada um desses métodos, foi calculado o tempo de execução utilizando a biblioteca time e para a inicialização das impressões de gráficos e histogramas, foi utilizada a biblioteca matplotlib.

C. Base de dados utilizada

Notificações de Síndrome Gripal (SG), realizadas no estado de São Paulo entre março e outubro, com 20.000 registros. Base de dados disponibilizada pelo Ministério da Saúde.

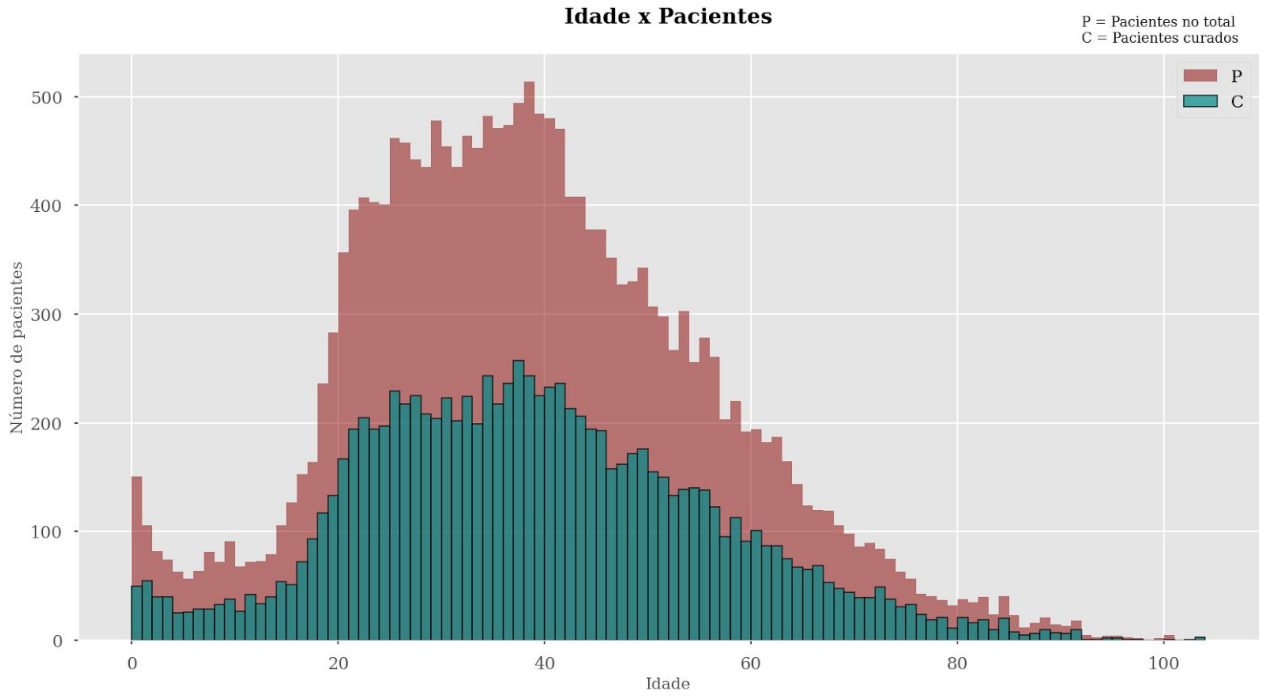

Figura 1 - Histograma contendo número de pacientes no geral e número de pacientes curados (eixo y) e idade (eixo x)

A Figura 1 mostra a comparação entre os pacientes no geral e os pacientes que constam como 'Curados' na base de dados disponibilizada pelo Ministério da Saúde.

Outro dado filtrado para análise foi a coluna de 'Sintomas' dos pacientes registrados. Foram analisados os pacientes assintomáticos e tabulados todos os seus dados para visualização, incluindo o id do paciente (chave primária), o dado de 'assintomático', o município, o dado que representa a conclusão ou não de um teste, o tipo de teste e o resultado do teste. No total, foram tabuladas 2151 colunas, ou seja, 2151 pacientes assintomáticos.

Uma outra informação disponibilizada nos registros dos pacientes foi a data e horário de notificação. Quando a coluna em questão é filtrada, tem-se uma lista com strings do tipo '2020-09-25T03:00:36.348Z', foi utilizada a função sorted( ), que é uma método estável de ordenação de strings, para ordenar esse *array*, depois disso, foram criados dois laços *for* para retirar a informação do horário e deixar apenas a data no padrão AAAA-MM-DD. Com isso, foi obtida a lista com as datas ordenadas.

Para melhor visualização dos dados, foi utilizada a função Counter( ) para contabilizar as ocorrências de cada data e, assim, gerar uma classe com o número de notificações que ocorreram em cada dia.

Depois disso, foi definida uma função que soma as notificações de um mês específico, para, assim, gerar um histograma que mostra, em barras, o número de notificações mensais. A função somarNotificacoes( ) é composta por um laço *for* que atribui à variável soma o incremento do elemento da notificação. Depois, foram construídas oito variáveis diferentes que recebem o valor de seu respectivo mês, para com ele, ser criada, posteriormente, uma lista com todos esses valores.

Para construir a tabela, foi criado um dicionário com as chaves mensais (março - outubro) e os seus respectivos valores. Além disso, foi criada uma função tabelarDicionario( ) que transformará esse dicionário em uma tabela para melhor visualização dos dados.

Mês	**Notificações**
Março	21
Abril	234
Maio	372
Junho	911
Julho	1502
Agosto	1255
Setembro	2075
Outubro	13722

Tabela 1 - Tabela de notificações mensais

Por fim, utilizando novamente a biblioteca do matplotlib, foi plotado um histograma mostrando esses números. Nele, é possível observar a relação entre os meses (eixo x) e a quantidade de notificações que foram registradas no estado de São Paulo (eixo y). Esse tipo de análise de dados é importante para observação de frequências e padrões em diversos tipos de dados.

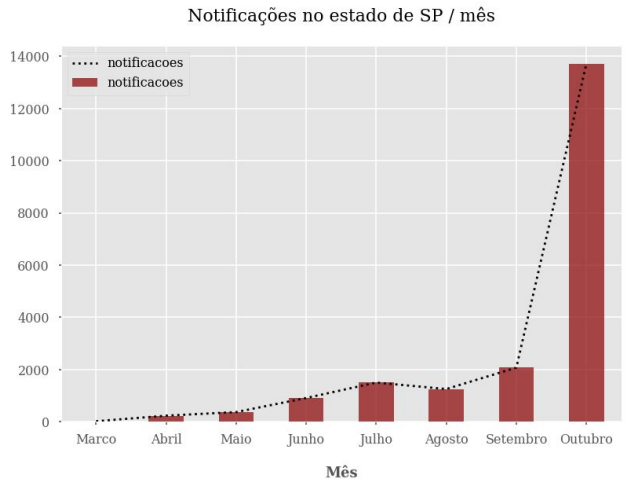

Figura 2 - Histograma com os valores de notificações mensais entre março e outubro de 2020.

Resultados e discussão

Um dos principais objetivos deste trabalho foi observar, em códigos, a execução de métodos de ordenação a partir de listas e *arrays* não ordenados. Neste caso, foi utilizado a lista com as idades de cada registro dos pacientes contidos na base de dados. Para construir a lista, foi filtrado do banco de dados apenas os valores da idade de cada linha, assim, construiu-se um array de 2000 posições.

Com essa lista não ordenada, foi realizada a ordenação com três diferentes métodos:

1. Inserção (Insertion Sort)
2. Contagem (Counting Sort)
3. Intercalação (Merge Sort)

	insertionSort	**countingSort**	**mergeSort**
1	O(n²)	O(n+k)	O(n log n)
2	14.73690	0.01498	0.11817

1. complexidade pior caso, 2. tempo de execução

Para realizar o estudo de complexidade de algoritmos, deve-se analisar os recursos necessários para a sua execução, isto é, cada operação executada pelo processadores, incluindo cálculos aritméticos lógicos, acesso à memória, comparações, etc. [2]

O método por inserção é um dos algoritmos mais simples para resolver problemas de ordenação. Nele, temos um laço *for* que percorrerá a lista completa e, dentro dele, um *while* que compara duas condições. Ao ser executado, realiza duas atribuições de valores. No caso do *insertion sort*, a sua complexidade no melhor caso é de O(n) e no pior caso O(n²).

Em *counting sort*, temos um algoritmo com complexidade O(n) que ordena um vetor considerando o seu elemento de valor máximo. Assim, considerando que a lista contém inteiros no intervalo 0 a k, ordena-se a lista simplesmente contando, para cada inteiro i, quantos elementos dessa lista são menores que ele [3]. Então, tem-se um laço que incrementa cada elemento do vetor, um laço que percorre de 0 a k e, dentro deste, outro laço percorrendo o vetor e realizando as comparações.

No caso do algoritmo por intercalação (Merge Sort), temos que o passo de divisão é executado em tempo constante, com apenas alguns cálculos de pontos. Existe um passo de conquista, que faz com que a função se torne recursiva. Essa recursão é realizada ordenando novamente as duas metades da lista. Esse algoritmo é composto por subproblemas, e conforme eles ficam menores, o número de subproblemas duplica-se em cada nível de recursão, porém o tempo de intercalação cai pela metade [4]. Assim, a função mergeSort( ) é executada em O(n log n).

Conclusões

A base de dados utilizada neste trabalho mostrou ser eficiente para a comparação de métodos de ordenação e análise de dados utilizando a biblioteca diversificada do Python 3.8. Com ela, foi possível analisar datas de notificações em hospitais em São Paulo, idade de pacientes e sintomas apresentados. Os métodos de ordenação analisados neste trabalho foram três, sendo os mais eficientes o Counting Sort e o Merge Sort. Conclui-se que o Insertion Sort, o método mais simples utilizado, pode ser menos eficiente quando atribuído a vetores grandes, como o caso de um vetor não ordenado de 20.000 elementos.

References

[1] Ministério da Saúde, "Notificações de Síndrome Gripal" <devopendatasus.saude.gov.br/dataset/casos-nacionais>. Acesso em: 19/12/2020.

[2] CHAIMOWICZ, Luiz. "Projeto e Análise de Algoritmos: Análise de Complexidade". UFMG

[3] C. C. de Souza, C. N. da Silva, O. Lee, P. J. de Rezende. "Complexidade de Algoritmos - v. 2. 1". UNICAMP.

[4] Khan Academy. "Análise do merge sort". <https://pt.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/analysis-of-merge-sort>. Acesso em: 21/12/2020.