RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
VOL. 3 NRO. 1 2022

O Uso da Análise de Dados Exploratória em Prol do Mapeamento Efetivo dos Casos de COVID-19 no Estado de São Paulo

Arthur Bomfim Neto
Graduando em Ciência da Computação – Uni-FACEF
arthur.bomfim.n@gmail.com

Patrícia de Castro Silva
Graduanda em Sistemas de Informação – Uni-FACEF
patykastros@gmail.com

Orientador: Prof. Me. Geraldo Henrique Neto
Mestre em Ciências com Ênfase em Informática Médica - FMRP-USP
geraldohenrique@alumni.usp.br

Resumo
A Covid-19 é uma doença respiratória infectocontagiosa causada pelo novo coronavírus, identificado no ano de 2019, denominado SARS-CoV-2, pertencente à família do vírus de mesmo nome, nomeado assim, pois ao ser analisado pelo microscópio, possui o formato de uma coroa. É uma doença potencialmente grave, altamente transmissível e que se espalhou por todo o mundo no ano de 2019. Este é um trabalho de caráter prático e exploratório, com embasamento bibliográfico, no qual foram feitas análises exploratórias em uma base de dados com os números epidemiológicos de toda população do estado de São Paulo, utilizando-se da linguagem Python, demonstrando visualmente através de gráficos, o impacto da doença em quantidade de casos e óbitos e as principais cidades acometidas pelo vírus.

Palavras-chave: Análise de Dados. Covid-19. Análise Exploratória.

Abstract
Covid-19 is an infection with a coronavirus contagious by the new disease, identified in the shape of a virus by the new virus, called SARS-CoV-2 a crown. It is a series of practical and known as epidemiological, highly transmissible numbers that are known throughout the world or are not practical in the work of 20 epidemiological and exploratory, with quality equivalent to exploratory in a database of the entire population of the state of São Paulo. Paulo, using the Python language, visually demonstrating through graphics, the impact of the disease in the number of cases and deaths and the main cities affected by the virus.

Keywords: *Data Analysis. Covid-19. Exploratory Analysis.*

Submissão: 15/10/2022. **Aprovação:** 22/10/2022.

1 Introdução

A Covid-19 (*Coronavirus Disease 2019*) é uma infecção respiratória causada pelo novo coronavírus, a Síndrome Respiratória Aguda Grave (SARS-CoV-2). O presente artigo tem como objetivo fazer uma análise exploratória de um *dataset* com

um conjunto de dados da pandemia no estado de São Paulo, disponibilizado pelo Seade (Fundação Sistema Estadual de Análise de Dados), uma fundação vinculada à Secretaria do Governo do Estado, que é um centro de referência nacional na produção e disseminação de análises e estatísticas socioeconômicas e demográficas, com 40 anos de existência, constituiu-se como uma fonte de dados segura e atualizada sobre o estado mais populoso do Brasil.

O estudo buscou analisar os dados e evidenciar quais os impactos da pandemia no estado, quais as cidades que mais sofreram com números de casos e óbitos e a sua desaceleração com a chegada da vacina para o combate à doença.

A linguagem Python foi utilizada para a concretização desse estudo, com a limpeza, integração, processamento e transformação da base, fazendo uso do *Jupyter Notebook,* como Ambiente de Desenvolvimento Integrado. Foram aplicados conceitos de estatística descritiva e análise exploratória de dados, a fim de que o leitor possa ser introduzido ao tema.

2 Referencial Teórico

Nessa seção serão abordados o tema principal da pesquisa, bem como a linguagem de programação, além das ferramentas e técnicas utilizadas para execução desse projeto.

2.1 O surgimento da Covid-19

Na atualidade, tem sido dada grande importância à Covid-19 (*Coronavirus Disease 2019*) que se trata de uma doença respiratória ocasionada pelo novo coronavírus SARS-CoV-2 ou Coronavírus da Síndrome Respiratória Aguda Grave 2 (SCHUCHMANN et al., 2020).

Os primeiros casos surgiram em outubro de 2019, mas a doença só foi identificada realmente, em dezembro do mesmo ano, na cidade de Wuhan, na província de Hubei na China. Segundo Nogueira (2020), o vírus possui similaridades com as infecções respiratórias causadas por SARS, um vírus que saltou de morcegos para pangolins e deste para os seres humanos por volta de 2002 e com o MERS-CoV que saltou de morcegos para camelos e deste para o homem em 2012, fato que leva a crer que o Sars-Cov2 possa ter características parecidas de transmissibilidade e origem evolutiva com estes vírus.

Em 11 de março de 2020, a Covid-19 foi declarado pela Organização Mundial da Saúde (OMS) como uma pandemia (SCHUCHMANN et al., 2020). Isso aconteceu justamente pela altíssima transmissibilidade e da grande difusão do vírus em todo globo terrestre. Nesse dia, o vírus já podia ser encontrado em 114 países com mais de 118 mil casos confirmados e 4.291 mortes.

A alta taxa de transmissão da doença, fez com que os casos logo disparassem ao redor de todo o mundo. Segundo o Ministério da Saúde do Brasil, de acordo com as evidências mais recentes, a transmissão ocorre da mesma forma que outros vírus respiratórios, de três maneiras:

Por contato: durante um aperto de mão seguido do toque nos olhos, nariz ou boca), ou com objetos e superfícies contaminados.

Por gotículas: através da exposição a gotículas respiratórias expelidas, contendo vírus, por uma pessoa infectada quando ela tosse ou espirra, principalmente quando ela se encontra a menos de 1 metro de distância da outra.

Por aerossol: por meio de gotículas respiratórias menores, contendo o vírus, que podem permanecer suspensas no ar.

O vírus causa desde um simples resfriado até doenças respiratórias mais graves.

> Os sintomas da Covid-19 são variados e podem ocorrer desde a forma mais branda até um acometimento grave com necessidade de internação hospitalar, sendo os principais: febre alta, tosse e dispneia. A infecção ainda pode acometer trato respiratório inferior e apresentar-se como pneumonia, por exemplo, corroborando para um caso mais grave. (SCHUCHMANN, 2020, p. 3558)

Mesmo não apresentando a mesma letalidade da SARS, a Covid-19 possui uma transmissibilidade superior, tornando-se assim, mais letal em números absolutos. Como seu período de incubação é alto, variando no princípio de 2 a 14 dias, além de alguns indivíduos serem assintomáticos, sua transmissão ficou cada vez mais difícil de ser controlada. (SCHUCHMANN et al., 2020).

2.2 Linguagem Python

O Python é uma linguagem de programação, com grande dinamismo e orientação a objetos, é muito utilizada para desenvolvimento de aplicações, mas também tem um grande valor para a ciência de dados, devido a sua grande versatilidade. Sua sintaxe é muito clara e simples e com enorme facilidade para ser aprendida.

A linguagem Python surgiu no final dos anos 80, na Holanda, por Guido van Rossum, que até hoje desempenha um papel importantíssimo no direcionamento de evolução da linguagem. Guido é conhecido dentre a grande comunidade de usuários como “Benevolent Dictator For Life”, ou seja, um ditador benevolente vitalício da linguagem.

Segundo Coelho (2007), a primeira versão disponibilizada foi a 0.9.0 e alcançou a 1.0 em 1994. Atualmente se encontra na versão 3.10.6. As principais características que a fizeram se tornar uma das linguagens mais utilizadas no mundo são: multiplataforma, portabilidade, *software* livre, extensibilidade, orientação a objeto, tipagem automática, tipagem forte, código legível, flexibilidade, operação com arquivos, uso interativo, entre inúmeras outras.

Python também é muito utilizada para a análise de dados e se destaca muito nesse universo.

Para análise de dados, processamento interativo e visualização de dados, Python inevitavelmente atrairá comparações com outras linguagens de programação comerciais e de código aberto, e com ferramentas amplamente utilizadas, como R, MATLAB, SAS, Stata e outras. Em conjunto com a robustez de Python para uma engenharia de *software* de propósito geral, é uma excelente opção como uma linguagem principal para a construção de aplicações de dados (MCKINNEY, 2018, *online*).

2.3 Bibliotecas disponíveis em Python

Um dos motivos do grande sucesso de Python na análise de dados, é que a linguagem possui uma enorme quantidade de bibliotecas voltadas ao processo aplicado à ciência de dados e a análise estatística. Essas bibliotecas possuem uma

série de funções, ferramentas e métodos para analisar e gerenciar dados, cada uma com um foco específico, visualização de dados, gerenciamento de dados e imagens, redes neurais, cálculos matemáticos e assim por diante. Na sequência, elencaremos as principais bibliotecas que dão subsídio para o processo de análise de dados na linguagem Python.

● Pandas: oferece uma grande facilidade na manipulação de dados na forma de tabelas e séries temporais. De acordo com McKinney (2018), o Pandas contém estruturas de dados e ferramentas, com intuito de facilitar e agilizar a limpeza, tratamento e análise de dados.

● NumPy: biblioteca que auxilia nas funções matemáticas de alto nível, em dados armazenados sob a forma de grandes *arrays* e matrizes, com ela é possível manipular esses dados com álgebra linear, processamento de números aleatórios e operações mais simples como fatiar, adicionar, multiplicar, indexar, remodelar e nivelar essas matrizes.

● Matplotlib: é voltada para visualização de dados e na plotagem de gráficos de barras, de linhas, histogramas, gráficos de dispersão, de erro, entre outros.

● Seaborn: também é uma biblioteca de visualização de dados com plotagem orientada a conjuntos. É uma interface para criação de gráficos estatísticos, visualmente mais bonitos e informativos que facilitam muito a exploração e compreensão, além de possibilitar a escolha de paletas de cores que podem revelar padrões nos dados.

2.4 Análise Exploratória de Dados

A Análise Exploratória de Dados, ou *EDA* é um trabalho de detetive numérico, de detetive de dados, de detetive gráfico, pois assim como um detetive criminal precisa de ferramentas para entender como ocorreu um crime, na análise exploratória de dados também é necessário o uso de ferramentas para coletar dados, formular hipóteses, avaliar padrões e identificar características que muitas vezes não podem ser notadas e que nos levam a conclusões sobre determinado assunto.

Uma *EDA* mal executada pode levar a erros, más interpretações de questões importantes e invalidação da análise se os dados forem incapazes de responder as questões-problema levantadas.

Segundo o blog *Dados ao Cubo* (2020), no processo de obtenção e tratamento de uma base são coletadas informações que ajudarão na compreensão dos dados e do resultado almejado. Cada característica da população amostrada no conjunto de dados é denominada de variável, que podem ser divididas em numéricas (quantitativas) e categóricas (qualitativas). As variáveis numéricas podem ser discretas (valores inteiros) ou contínuas (qualquer valor, incluindo números reais). Já as varáveis categóricas podem ser do tipo nominais (quando não podem ser ordenadas de alguma maneira) ou ordinais (quando podem ser ordenadas).

No presente estudo, foram analisadas variáveis quantitativas de medidas de posição, como:

- Média: soma de todos os dados da amostra, dividido pela quantidade de amostras.
- Mediana: valor que deixa 50% das observações à sua esquerda.
- Moda: é o valor que aparece com maior frequência em um conjunto de dados.
- Máximo: valor máximo de um conjunto de dados.

- Mínimo: valor mínimo de um conjunto de dados.

E de medidas de dispersão, como:

- Variância: medida de dispersão dos dados, mede o quão afastados da média estão os dados.
- Desvio Padrão: mede a variabilidade independentemente do número de observações e com a mesma unidade de medida da média.

O exame gráfico dos dados contou com o *boxplot*, que registra, avalia e compara o formato, tendência central e variabilidade de distribuição das amostras em busca de possíveis *outliers*.

3 Resultados da Análise de Dados

Essa análise de dados tem como objetivo principal evidenciar os principais impactos da pandemia no estado de São Paulo, as cidades mais atingidas pelos casos e pelas mortes, as curvas de contágio e a aceleração e desaceleração da doença em função do tempo.

3.1 Preparação da Base de Dados

O *dataset* utilizado foi obtido no Portal Seade do Governo do Estado de São Paulo e contém dados referentes à pandemia em todas as cidades do estado.

Iniciamos a análise importando as bibliotecas: Pandas, Seaborn, Matplotlib e Numpy, que foram utilizadas no decorrer da análise exploratória utilizada no presente trabalho, conforme apresentado na Figura 1.

Figura 1 — Importação das bibliotecas

```
In [1]: # Importar Bibliotecas
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        import numpy as np
```

Fonte: os autores.

Em seguida foi feito a leitura dessa base de dados, intitulada *dados_covid_brazil.csv.* Utilizamos a função da biblioteca pandas: pd.read_csv que é específica para arquivos com a extensão csv, *comma-separated values* (valores separados por vírgula). É um processo que armazena os dados em formato de tabela, em um *dataframe*, o qual nomeamos como “df_total” (Figura 2).

Figura 2 — Leitura dos dados

```
In [2]: # Leitura do arquivo
        df_total = pd.read_csv('dados_covid_brazil.csv')
```

Fonte: os autores.

Utilizamos a função *head*, que retorna os registros do *dataframe* criado, para visualização das primeiras ocorrências.

A Figura 3 ilustra o comando e o resultado obtido.

Figura 3 — Visualização das 5 primeiras entradas

```
In [3]: # Visualizar as 5 primeiras entradas
        df_total.head()
```

Out[3]:

	nome_munic	codigo_ibge	dia	mes	datahora	casos	casos_novos	casos_pc	casos_mm7d	obitos	...	nome_drs	cod_drs	pop	pop_60	area	map_l
0	Adamantina	3500105	25	2	2020-02-25	0	0	0.0	0.0	0	...	Marília	5	33894	7398	41199	
1	Adolfo	3500204	25	2	2020-02-25	0	0	0.0	0.0	0	...	São José do Rio Preto	15	3447	761	21106	
2	Aguaí	3500303	25	2	2020-02-25	0	0	0.0	0.0	0	...	São João da Boa Vista	14	35608	5245	47455	
3	Águas da Prata	3500402	25	2	2020-02-25	0	0	0.0	0.0	0	...	São João da Boa Vista	14	7797	1729	14267	
4	Águas de Lindóia	3500501	25	2	2020-02-25	0	0	0.0	0.0	0	...	Campinas	3	18374	3275	6013	

5 rows × 26 columns

Fonte: os autores.

Podemos encontrar as dimensões do *dataframe* utilizando a função *shape,* que retorna à quantidade de linhas e colunas do objeto. O *dataframe* df_total contém 572.356 linhas e 26 colunas, conforme visualizado na Figura 4.

Figura 4 — Tamanho do dataframe

```
In [4]: # Verificar o tamanho do DataFrame
        print('O DataFrame possui {} entradas e {} colunas.'.format(df_total.shape[0], df_total.shape[1]))

O DataFrame possui 572356 entradas e 26 colunas.
```

Fonte: os autores.

As colunas apresentam informações como: nome do município, código do IBGE, dia, mês, data completa, quantidade de casos, quantidade de casos novos, casos totais por 100 mil habitantes, média móvel dos últimos 7 dias dos novos casos, quantidade de óbitos, quantidade de óbitos novos, óbitos totais por 100 mil habitantes, média móvel dos últimos 7 dias dos óbitos novos, letalidade, entre outras não tão relevantes para essa análise, apresentados pela Figura 5.

Figura 5 — Exibição dos nomes das colunas

```
In [5]: # Verificar os nomes das colunas
        df_total.columns

Out[5]: Index(['nome_munic', 'codigo_ibge', 'dia', 'mes', 'datahora', 'casos',
               'casos_novos', 'casos_pc', 'casos_mm7d', 'obitos', 'obitos_novos',
               'obitos_pc', 'obitos_mm7d', 'letalidade', 'nome_ra', 'cod_ra',
               'nome_drs', 'cod_drs', 'pop', 'pop_60', 'area', 'map_leg', 'map_leg_s',
               'latitude', 'longitude', 'semana_epidem'],
              dtype='object')
```

Fonte: os autores.

Se a base contiver dados nulos, pode-se comprometer a construção dessa análise. Por isso é necessária a verificação do percentual de valores desse tipo. Para isso utilizamos um comando que demonstra exatamente quais colunas são afetadas por essa questão (Figura 6). No *dataframe* df_total é possível notar a presença de dados ausentes, analisamos cada uma das variáveis que possuem uma grande quantidade de valores faltantes e definir o tratamento mais adequado.

Analisando os dados ausentes, é notável que: map_leg_s (código da legenda para mapa), mapa_leg (rótulo da legenda para mapa), nome_drs (nome do departamento regional de saúde) e nome_ra (nome da região administrativa) são colunas que não foram utilizadas para nossa análise e por isso não receberam tratamento.

Figura 6 — Verificando se existem registros nulos ou ausentes

```
In [6]: # Checar a porcentagem de valores ausentes
        (df_total.isnull().sum()/ df_total.shape[0]).sort_values(ascending=False)

Out[6]: map_leg_s        0.001548
        map_leg          0.001548
        nome_drs         0.001548
        nome_ra          0.001548
```

Fonte: os autores.

Essa análise dependerá muito da variável do tempo, pois trata-se também de uma análise de série temporal e, por isso, é muito importante que o tipo de dado da coluna “datahora” seja do tipo correto: data. A Figura 7 exibe o comando e o retorno com o tipo dessa variável, que nesse caso é do tipo objeto.

Figura 7 — Verificando o tipo de dado da coluna “datahora”

```
In [7]: # Verificar os tipo da variável "DATA"
        dict(df_total.dtypes) ['datahora']

Out[7]: dtype('O')
```

Fonte: os autores.

Para uma análise com uma eficiência maior, iremos converter a coluna de *object* para *datetime*, processo que pode ser visualizado pela Figura 8.

Figura 8 — Convertendo a coluna data

```
In [8]: # Transformar coluna "datahora" no formato Datetime
        df_total.datahora = pd.to_datetime(df_total.datahora)
```

Fonte: os autores.

3.2 Exploração da Base de Dados

Como vimos anteriormente, o *dataframe* é composto por uma grande quantidade de linhas e colunas. Por isso, agrupamos a soma das informações mais importantes para uma melhor performance da exploração, em um novo *dataframe* denominado “df_data”, utilizando a função *groupby*, conforme representado pela Figura 9.

Figura 9 — Agrupando a soma de colunas

```
In [9]: # Utilização da função groupby, agrupando a soma das colunas e criando um novo data-frame (df_data)

        # Exibição das últimas 5 ocorrências
        df_data=df_total.groupby('datahora')['casos_novos','casos','obitos_novos', 'obitos']
        .agg({'casos_novos':'sum','casos':'sum','obitos_novos':'sum','obitos':'sum'}).reset_index()

        df_data.tail(5)
```

Out[9]:

	datahora	casos_novos	casos	obitos_novos	obitos
881	2022-07-25	652	5883611	11	172555
882	2022-07-26	7152	5890763	99	172654
883	2022-07-27	7096	5897859	73	172727
884	2022-07-28	6986	5904845	89	172816
885	2022-07-29	6707	5911552	71	172887

Fonte: os autores.

O *dataset* utilizado não conta com uma informação fundamental para entendermos o impacto da doença no decorrer do tempo, que é a quantidade de casos e óbitos acumulados por dia. Esse dado nos permite identificar padrões de crescimento ou diminuição das ocorrências.

Uma análise de dados eficaz conta histórias e traz respostas relevantes. Muitas das vezes, as bases analisadas não possuem informações que podem ser de extrema importância para responder essas perguntas. Com isso, é preciso tratar essa base, através de funções que possam nos fornecer esses dados, o comando que demonstra esse tratamento pode ser visto na Figura 10.

Figura 10 — Criação das colunas de casos e óbitos acumulados.

```
In [10]: # Criação de colunas com as informações acumuladas por data

obitosAcumulados = []
casosAcumulados = []
obitos = 0
casos = 0

for i, row in df_data.iterrows():
    casos += row['casos_novos']
    casosAcumulados.append(casos)

df_data['casos_acumulados'] =  casosAcumulados

for i, row in df_data.iterrows():
    obitos += row['obitos_novos']
    obitosAcumulados.append(obitos)

df_data['obitos_acumulados'] =  obitosAcumulados

# As 10 últimas ocorrências

df_data.tail(10)
```

Out[10]:

	datahora	casos_novos	casos	obitos_novos	obitos	casos_acumulados	obitos_acumulados
876	2022-07-20	8303	5861263	92	172277	5861263	172277
877	2022-07-21	6608	5867871	87	172364	5867871	172364
878	2022-07-22	6745	5874616	95	172459	5874616	172459
879	2022-07-23	6117	5880733	77	172536	5880733	172536
880	2022-07-24	2226	5882959	8	172544	5882959	172544
881	2022-07-25	652	5883611	11	172555	5883611	172555
882	2022-07-26	7152	5890763	99	172654	5890763	172654
883	2022-07-27	7096	5897859	73	172727	5897859	172727
884	2022-07-28	6986	5904845	89	172816	5904845	172816
885	2022-07-29	6707	5911552	71	172887	5911552	172887

Fonte: os autores.

Na sequência (Figura 11), foi possível extrairmos medidas importantes em uma análise exploratória de dados, a citar: variância, desvio padrão, média, mediana, moda, valor máximo e mínimo de casos e óbitos.

Figura 11 — Utilizando funções estatísticas com os casos novos

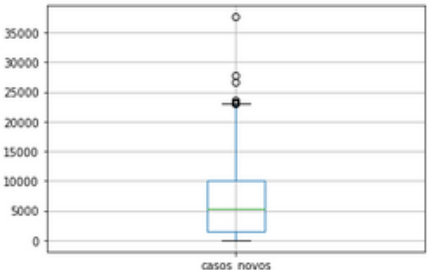
```
In [11]: # para variancia e desvio padrao ddof=0(populacional)  ddof=1(amostral)

variancia = df_data['casos_novos'].var(ddof=0)
desvPd = df_data['casos_novos'].std(ddof=0)
media = df_data['casos_novos'].mean()
mediana = df_data['casos_novos'].median()
maximo = df_data['casos_novos'].max()
minimo = df_data['casos_novos'].min()
moda = df_data['casos_novos'].mode()

print("Variância: ", round(variancia,2) )
print("Desvio padrão: ", round(desvPd,2) )
print("Media: ", round(media,2 ))
print("Mediana: ", mediana)
print("Maximo de casos no dia: ",maximo)
print("Minimo de casos no dia: ",minimo)
print("Moda",moda)


Variância:  35615210.29
Desvio padrão:  5967.85
Media:  6672.18
Mediana:  5325.0
Maximo de casos no dia:  37611
Minimo de casos no dia:  -3
Moda 0    0
dtype: int64
```

Fonte: os autores.

Dessa forma, foi possível visualizar esses dados, utilizando o gráfico de *boxplot,* conforme mostrado pela Figura 12. Através dele os dados são apresentados visualmente, mostrando os valores mínimos e máximos, mediana, quartis e evidenciando os valores discrepantes em relação à média.

Figura 12 — Boxplot de casos novos

```
In [12]: # Detecção de Outliers de casos por dia

         df_data.boxplot(column='casos_novos')
         plt.show()
```



Fonte: os autores.

Identificamos que as dispersões ocorrem acima dos 25.000 casos novos por dia, e, esmiuçando a análise, foi factível evidenciar que os dias 01/04/2021, 18/06/2021 e 03/02/2022 foram as datas que ocorreram os principais *outliers* (Figura 13).

Figura 13 — Verificando os principais outliers

```
In [13]: # os 3 principais outliers

         df_data[df_data.casos_novos > 25000]
```

Out[13]:

	datahora	casos_novos	casos	obitos_novos	obitos	casos_acumulados	obitos_acumulados
401	2021-04-01	26567	2496416	1082	75734	2496416	75734
479	2021-06-18	27706	3552727	714	121238	3552727	121238
709	2022-02-03	37611	4704552	349	158872	4704552	158872

Fonte: os autores.

Na Figura 14, podemos observar os resultados das mesmas medidas, porém com o número de óbitos.

Figura 14 — Utilizando funções estatísticas com os óbitos novos

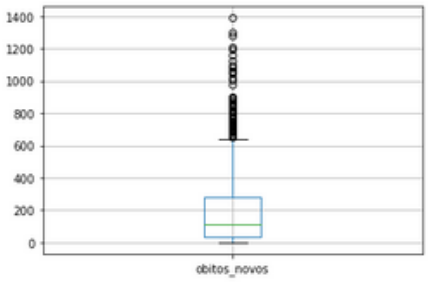
```
In [14]: # Para variância e desvio padrao ddof=0(populacional)  ddof=1(amostral)

         variancia = df_data['obitos_novos'].var(ddof=0)
         desvPd = df_data['obitos_novos'].std(ddof=0)
         media = df_data['obitos_novos'].mean()
         mediana = df_data['obitos_novos'].median()
         maximo = df_data['obitos_novos'].max()
         minimo = df_data['obitos_novos'].min()
         moda = df_data['obitos_novos'].mode()


         print("Variância: ", round(variancia,2) )
         print("Desvio padrão: ", round(desvPd,2) )
         print("Media: ", round(media,2 ))
         print("Mediana: ", mediana)
         print("Maximo de casos no dia: ",maximo)
         print("Minimo de casos no dia: ",minimo)
         print("Moda",moda)

Variância:  52281.92
Desvio padrão:  228.65
Media:  195.13
Mediana:  111.0
Maximo de casos no dia:  1389
Minimo de casos no dia:  -1
Moda 0    0
dtype: int64
```

Fonte: os autores.

O *boxplot* do número de óbitos, mostra as medidas e *outliers* (ver Figura 15).

Figura 15 — Boxplot de óbitos novos

```
In [15]: #Detecção de Outliers de Obitos por dia

         df_data.boxplot(column='obitos_novos')
         plt.show()
```



Fonte: os autores.

3.3 Análise dos Dados

Com os dados preparados e tratados, é possível iniciar uma análise mais profunda sobre os impactos da pandemia no estado de São Paulo. Para isso precisamos descobrir qual a data da última entrada do *dataframe*, pois é através dela que partiremos todas as outras averiguações. Através da função *max*, percebemos que o último registro data de 29 de julho de 2022 (Figura 16).

Figura 16 — Verificando data do último registro

```
In [16]: # Checar a data da última entrada do DataFrame
         df_total.datahora.max()

Out[16]: '2022-07-29'
```

Fonte: Os autores.

Munidos dessa informação utilizamos o comando *loc* para retornar as 5 cidades com a maior quantidade de casos (Figura 17).

Figura 17 — As 5 cidades com maior quantidade de casos

```
In [19]: # Localizar quais cidades possuem mais casos para data mais atual '2022-07-29'
         df_total.loc[~df_total.nome_munic.isin(nome_munic)][df_total.datahora == '2022-07-29']
         .sort_values(by='casos', ascending=False)[0:5]
```

Out[19]:

	nome_munic	codigo_ibge	dia	mes	datahora	casos	casos_novos	casos_pc	casos_mm7d	obitos	...	nome_drs	cod_drs	pop	pop_60
571387	São Paulo	3550308	29	7	2022-07-29	1104159	758	9302.364179	499.571429	43387	...	Grande São Paulo	10	11869660	1853286
570933	Campinas	3509502	29	7	2022-07-29	185387	88	15770.892581	147.714286	5237	...	Campinas	3	1175501	192796
571381	São José do Rio Preto	3549805	29	7	2022-07-29	135497	7	30249.997767	89.285714	3155	...	São José do Rio Preto	15	447924	79391
571382	São José dos Campos	3549904	29	7	2022-07-29	124665	129	17542.292030	80.285714	2267	...	Taubaté	17	710654	103840
571406	Sorocaba	3552205	29	7	2022-07-29	104603	30	15883.908058	37.285714	3180	...	Sorocaba	6	658547	96498

5 rows × 26 columns

Fonte: os autores.

E as cidades do estado que mais sofreram com o número de óbitos (Figura 18).

Figura 18 — As 5 cidades com maior quantidade de óbitos

```
In [20]: # Localizar quais cidades possuem mais mortes para data mais atual '2022-07-29'
df_total.loc[~df_total.nome_munic.isin(nome_munic)][df_total.datahora == '2022-07-29']
.sort_values(by='obitos', ascending=False)[0:5]
```

Out[20]:

	nome_munic	codigo_ibge	dia	mes	datahora	casos	casos_novos	casos_pc	casos_mm7d	obitos	...	nome_drs	cod_drs	pop	pop_60
571387	São Paulo	3550308	29	7	2022-07-29	1104159	758	9302.364179	499.571429	43387	...	Grande São Paulo	10	11869660	1853286
571037	Guarulhos	3518800	29	7	2022-07-29	84029	13	6218.497345	11.571429	5454	...	Grande São Paulo	10	1351275	162662
570933	Campinas	3509502	29	7	2022-07-29	185387	88	15770.892581	147.714286	5237	...	Campinas	3	1175501	192796
571369	São Bernardo do Campo	3548708	29	7	2022-07-29	76360	7	9402.945008	12.000000	3733	...	Grande São Paulo	10	812086	122274
571359	Santo André	3547809	29	7	2022-07-29	91822	6	13233.371813	18.000000	3458	...	Grande São Paulo	10	693867	123293

5 rows × 26 columns

Fonte: os autores.

Assim, foi possível criarmos um *dataframe* apenas com essas cidades impactadas, gravando todos os seus dados, utilizando-se da busca pelo código do IBGE como exemplificado na Figura 19.

Figura 19 — Criando um dataframe com as 5 cidades com mais casos e mais mortes

```
In [55]: # Localizando os dados sobre as 5 cidades
SP = df_total.loc[df_total.codigo_ibge == 3550308]
CAMP = df_total.loc[df_total.codigo_ibge == 3509502]
SJRP = df_total.loc[df_total.codigo_ibge == 3549805]
SJC = df_total.loc[df_total.codigo_ibge == 3549904]
SOR = df_total.loc[df_total.codigo_ibge == 3552205]
GRU = df_total.loc[df_total.codigo_ibge == 3518800]
SBC = df_total.loc[df_total.codigo_ibge == 3548708]
STA = df_total.loc[df_total.codigo_ibge == 3547809]

# Criando DataFrame com os Países selecionados
cidades_grafico_casos = SP.append(CAMP).append(SJRP).append(SJC).append(SOR)
cidades_grafico_mortes = SP.append(GRU).append(CAMP).append(SBC).append(STA)
```

Fonte: os autores.

Na sequência, foi possível armazenar esses dados em novas variáveis, buscando apenas as colunas de nosso interesse: data, cidade, casos e óbitos (Figura 20).

Figura 20 — Armazenando somente os dados importantes das 5 cidades impactadas

```
In [22]: # Localizando Variáveis na data mais recente
total_casos = cidades_grafico_casos.loc[cidades_grafico_casos.datahora == '2022-07-29',['datahora', 'nome_munic', 'casos']]
.sort_values(by='casos', ascending=False)

total_mortes = cidades_grafico_mortes.loc[cidades_grafico_mortes.datahora == '2022-07-29',['datahora', 'nome_munic', 'obitos']]
.sort_values(by='obitos', ascending=False)
```

Fonte: os autores.

Finalizando o processo, foi possível plotar e visualizar esses dados através de gráficos, por meio das bibliotecas Seaborn e Matplotlib, que nos mostrou o número de casos e mortes dessas 5 cidades mais atingidas. Com a utilização da função *subplot*, conseguimos visualizar vários gráficos em uma mesma linha ou coluna visual, nesse caso temos 2 na mesma linha, conforme visualizado na Figura 21.

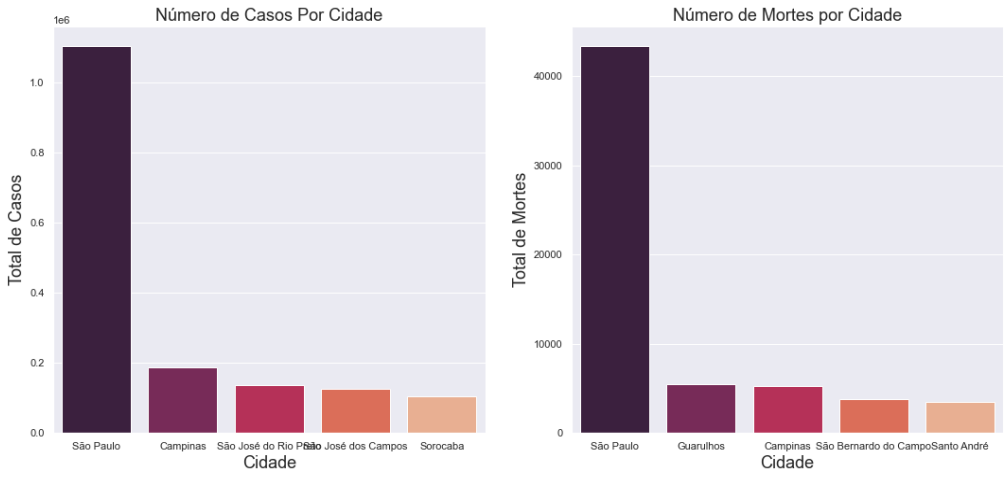
Figura 21 — Comando de plotagem de gráfico de barras

```
In [23]: # Plotando os gráficos de barras
fig, ax = plt.subplots(nrows=1 , ncols=2, figsize=(18,8))

# Definindo estilo dos Gráficos para o estilo do Seaborn
sns.set()

# Gráfico 1
sns.barplot(x='nome_munic', y=total_casos['casos'], data=cidades_grafico_casos, ax=ax[0], palette='rocket')
ax[0].set_title('Número de Casos Por Cidade',fontsize=18)
ax[0].set_ylabel('Total de Casos',fontsize=18)
ax[0].set_xlabel('Cidade',fontsize=18)


# Gráfico 2
sns.barplot(x='nome_munic', y=total_mortes['obitos'], data=cidades_grafico_mortes, ax=ax[1], palette='rocket')
ax[1].set_title('Número de Mortes por Cidade',fontsize=18)
ax[1].set_ylabel('Total de Mortes',fontsize=18)
ax[1].set_xlabel('Cidade',fontsize=18)

#plt.tick_params(labelsize=25)
plt.tight_layout();


# Salvar Imagem
fig.savefig('barrasCovidCidade.png')
```

Fonte: os autores.

A Figura 22 apresentou essas 5 cidades de forma bem simples, através de um gráfico de barras gerado pelo comando anterior.

Figura 22 — Gráfico de Barras



Fonte: os autores.

Outra questão de grande relevância foi a possibilidade de analisar os impactos em relação ao tempo. Com ela foi possível ver as curvas de contágio e de morte provocadas pela doença. Isso pode ser visto através do comando na Figura 23.

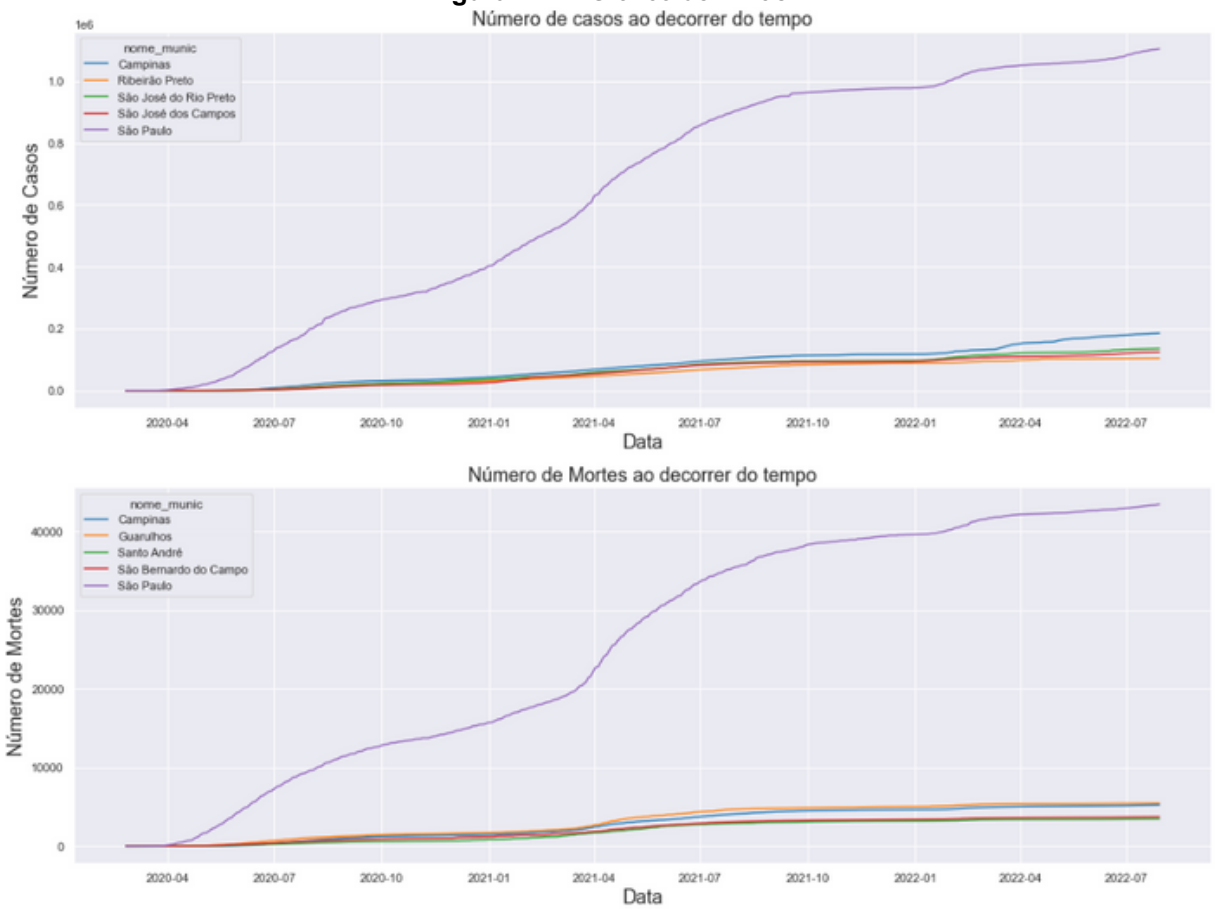
Figura 23 — Comando de plotagem de gráfico de linhas

```
In [14]: # Plotando os Gráficos de Linha
fig , ax = plt.subplots(nrows=2, ncols=1, figsize=(16,12))

# Gráfico 1
sns.lineplot(data=df_total,x=df_total['datahora'], y=df_total['casos'] , hue=cidades_grafico_casos['nome_munic'], palette='tab10'
ax[0].set_title('Número de casos ao decorrer do tempo',fontsize=18)
ax[0].set_xlabel('Data',fontsize=18)
ax[0].set_ylabel('Número de Casos',fontsize=18)


# Gráfico 3
sns.lineplot(data=df_total, x=df_total['datahora'], y=df_total['obitos'], hue=cidades_grafico_mortes['nome_munic'], palette='tab1
ax[1].set_title('Número de Mortes ao decorrer do tempo',fontsize=18)
ax[1].set_xlabel('Data',fontsize=18)
ax[1].set_ylabel('Número de Mortes',fontsize=18)


plt.tight_layout();

# Salvar Imagem
fig.savefig('linhasCovidSP.png')
```

Fonte: os autores.

O gráfico de linhas é muito utilizado para visualização de tendências e na Figura 24 é possível observar como se comportou a curva de casos e mortes nessas cidades, no decorrer do tempo.

Figura 24 — Gráfico de linhas



Fonte: os autores.

Dando sequência ao processo, foi possível descobrir a data do primeiro óbito no estado de São Paulo, que ocorreu na capital paulista no dia 17 de março de 2020 (Figura 25).

Figura 25 — Comando para identificar a primeira morte

```
In [9]: # Localizar a data da primeira morte no Estado
        df_total.loc[df_total['obitos_novos'] == 1.0].head(1)
```

Out[9]:

	nome_munic	codigo_ibge	dia	mes	datahora	casos	casos_novos	casos_pc	casos_mm7d	obitos	...	nome_drs	cod_drs	pop	pop_60	area
14107	São Paulo	3550308	17	3	2020-03-17	156	11	1.314275	19.714286	1	...	Grande São Paulo	10	11869660	1853286	152111

Fonte: os autores.

O primeiro caso foi registrado 2 semanas antes, em 28 de fevereiro de 2020 (Figura 26).

Figura 26 — Comando para identificar o primeiro caso

```
In [10]: # Localizar a data do primeiro caso no Estado
         df_total.loc[df_total['casos_novos'] != 0.0].head(1)
```

Out[10]:

	nome_munic	codigo_ibge	dia	mes	datahora	casos	casos_novos	casos_pc	casos_mm7d	obitos	...	nome_drs	cod_drs	pop	pop_60	area
2497	São Paulo	3550308	28	2	2020-02-28	2	1	0.01685	0.0	0	...	Grande São Paulo	10	11869660	1853286	152111

1 rows × 26 columns

Fonte: os autores.

Com o pacote *date,* foi possível realizar cálculos com colunas do formato de data e identificar por exemplo, o intervalo de tempo entre o primeiro caso e a primeira morte no estado, é o que podemos visualizar na Figura 27.

Figura 27 — Identificando o intervalo de dias entra o primeiro caso e primeira morte.

```
In [11]: # importar pacote nescessário para identificar quantos dias se passaram para registrar a primeira morte após o primeiro caso
         from datetime import date

         # Calculando quantos dias entre o primeiro caso registrado e a primeira morte
         dias = df_total.datahora.loc[14107] - df_total.datahora.loc[2497]
         print('Foram {} dias entre o primeiro caso e a primeira morte no estado de São Paulo.'.format(dias.days))

Foram 18 dias entre o primeiro caso e a primeira morte no estado de São Paulo.
```

Fonte: os autores.

A letalidade da doença é uma variável que traz um aspecto muito interessante para a análise, pois com ela se identifica qual o percentual de óbitos em relação a quantidade de casos. Assim utilizamos o comando *loc* para relacionar as 5 cidades com a maior letalidade, conforme Figura 28.

Figura 28 — Cidades com maior letalidade

```
In [29]: # Localizar quais cidades possuem maior letalidade para a data mais atual '2022-07-29'
         df_total.loc[~df_total.nome_munic.isin(nome_munic)][df_total.datahora == '2022-07-29']
         .sort_values(by='letalidade', ascending=False)[0:5]
```

Out[29]:

	nome_munic	codigo_ibge	dia	mes	datahora	casos	casos_novos	casos_pc	casos_mm7d	obitos	...	nome_drs	cod_drs	pop	pop_60	area
571396	Sarutaiá	3551207	29	7	2022-07-29	146	0	4004.388371	0.000000	20	...	Bauru	12	3646	643	14161
571325	Sabino	3544608	29	7	2022-07-29	303	0	5541.331383	0.000000	30	...	Bauru	12	5468	1103	30529
571329	Salesópolis	3545001	29	7	2022-07-29	709	0	4210.713862	0.428571	62	...	Grande São Paulo	10	16838	2512	425
570991	Embaúba	3514957	29	7	2022-07-29	129	0	5354.919054	0.000000	11	...	São José do Rio Preto	15	2409	461	8313
571045	Iaras	3519253	29	7	2022-07-29	300	0	4351.610096	0.000000	24	...	Bauru	12	6894	888	40138

Fonte: os autores.

Em seguida plotamos o gráfico comparando com a cidade de São Paulo que para nível de informação, identificamos que foi a mais atingida pela pandemia (Figura 29).

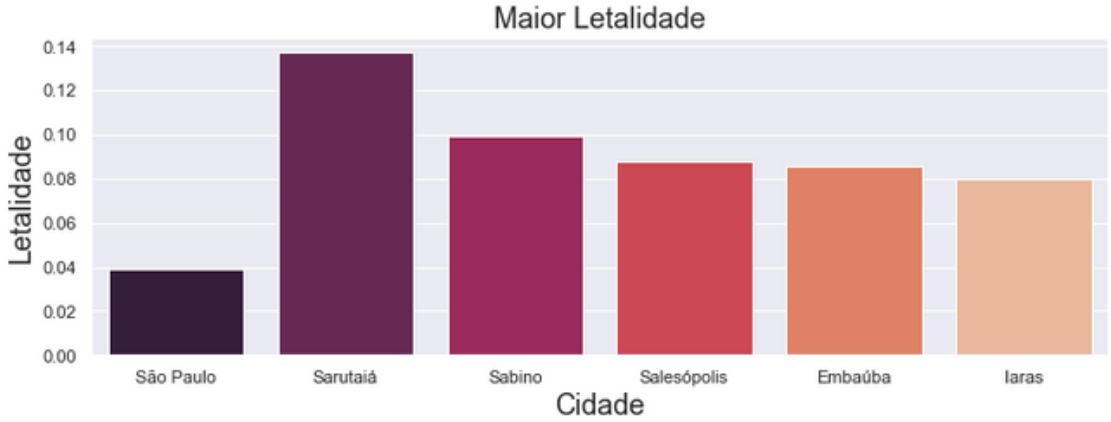
Figura 29 — Comando para plotagem do gráfico de maior letalidade

```
In [31]: # Localizando os dados sobre as 5 cidades
SAR = df_total.loc[df_total.codigo_ibge == 3551207]
SAB = df_total.loc[df_total.codigo_ibge == 3544608]
SAL = df_total.loc[df_total.codigo_ibge == 3545001]
EMB = df_total.loc[df_total.codigo_ibge == 3514957]
IAR = df_total.loc[df_total.codigo_ibge == 3519253]

# Criando DataFrame com as cidades com maior letalidade
cidades_grafico_maior_letalidade = SP.append(SAR).append(SAB).append(SAL).append(EMB).append(IAR)

# Localizando Variáveis na data mais recente
total_maior_letalidade = cidades_grafico_maior_letalidade.loc
[cidades_grafico_maior_letalidade.datahora == '2022-07-29',['datahora', 'nome_munic', 'letalidade']].
sort_values(by='letalidade', ascending=False)

# Plotando os gráficos de barras
sns.set()

# Gráfico
plt.figure(figsize = (10,4))
fig = sns.barplot(x='nome_munic', y=total_maior_letalidade['letalidade'], data=cidades_grafico_maior_letalidade,
                  palette='rocket')
fig.set_title('Maior Letalidade',fontsize=18)
fig.set_ylabel('Letalidade',fontsize=18)
fig.set_xlabel('Cidade',fontsize=18)

#plt.tick_params(labelsize=25)
plt.tight_layout();
```

Fonte: os autores.

Através do gráfico da Figura 30 foi possível observar que São Paulo, mesmo sendo a cidade mais acometida pela quantidade de casos, tem seu índice de letalidade (4%) bem menor que as cidades observadas, Sarutaiá por exemplo possui 13,6% de mortes em relação ao número de casos.

Figura 30 — Gráfico de letalidade



Fonte: os autores.

Partindo do ponto que a distribuição da nossa base em questão possui um viés, ou seja, uma das extremidades elevadas, medidas de regressão e correlação podem ser influenciadas pelos *outliers*. Aplicando a transformada logarítmica, podemos reduzir esse viés.

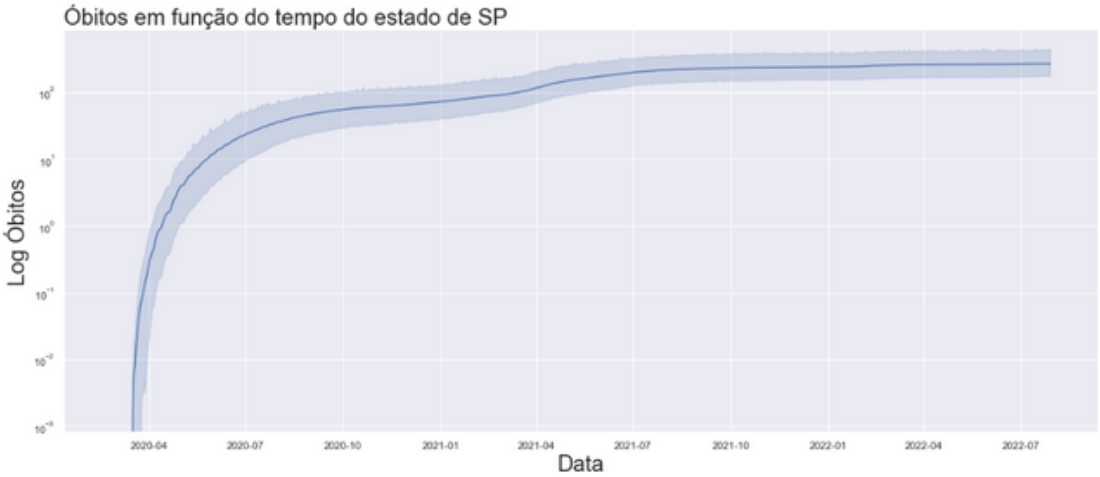
Para analisar o crescimento exponencial dos casos e óbitos, precisamos entender primeiramente como funciona a relação entre exponencial e logaritmo.

No caso da relação exponencial, se a elevarmos a algum valor, que é a taxa de crescimento da Covid-19, quando aplicarmos o logaritmo natural, temos a taxa de

crescimento. Então aplicamos o logaritmo nos óbitos e assim temos a curva e o comportamento dessa curva demonstrou que ela já chegou em seu platô, a taxa de crescimento não está mais em ascensão (Figura 31).

Figura 31 — Transformada logarítmica de óbitos

```
In [121]: plt.figure (figsize = (20,8))
          fig = sns.lineplot(x= 'datahora', y= 'obitos', data=df_total)
          fig.set_title('Óbitos em função do tempo do estado de SP', loc= 'left', fontsize= 24)
          fig.set_xlabel ('Data',fontsize= 24)
          fig.set_ylabel ('Log Óbitos',fontsize= 24)
          fig.set_yscale('log')
```



Fonte: os autores.

Para plotar gráficos de barras e assim facilitar a análise, elaboramos uma função personalizada para esse propósito, essa função pode ser vista na Figura 32.

Figura 32 — Função para plotagem de gráficos de barras.

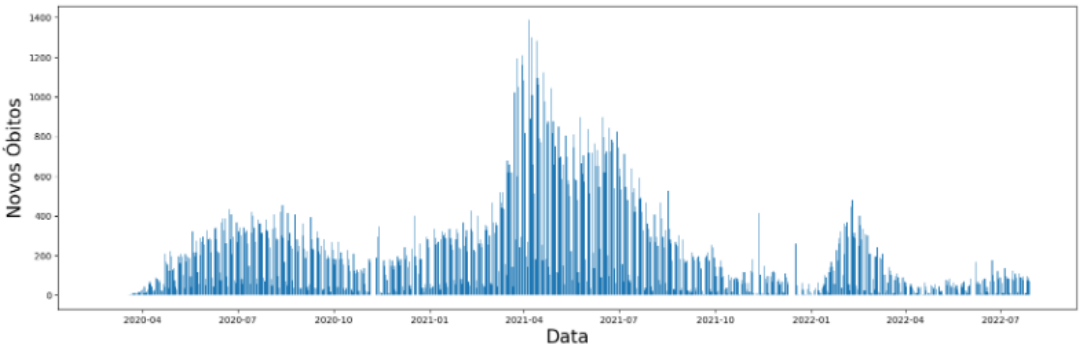
```
In [18]: #função para plotar barra
         def plotar_barra(titulo, xlabel, ylabel, x, y, dataset):
             plt.figure(figsize=(24,8))
             ax = plt.bar(dataset[x], dataset[y])
             plt.xlabel(xlabel, fontsize=20)
             plt.ylabel(ylabel, fontsize=20, rotation=90)
```

Fonte: os autores.

Munidos da função desenvolvida anteriormente, foi feita uma nova chamada para demonstrar o aumento de óbitos conforme a linha do tempo (Figura 33).

Figura 33 — Gráfico de novos óbitos por dia.

```
In [21]: #Novos óbitos em SP
         plotar_barra('Novos óbitos em SP','Data','Novos Óbitos','datahora','obitos_novos', df_data)
```



Fonte: os autores.

Nosso gráfico apresentou uma oscilação muito relevante nos números de óbitos, alguns dias com números muito altos, outros dias com números muito baixos, isso demonstrou que tivemos acelerações, ou seja, diferenças razoavelmente grandes entre dias. Sendo assim, calculamos a aceleração dos novos óbitos, ou seja, a velocidade com que a taxa cresce, através da função *diff*. (Ver Figura 34).

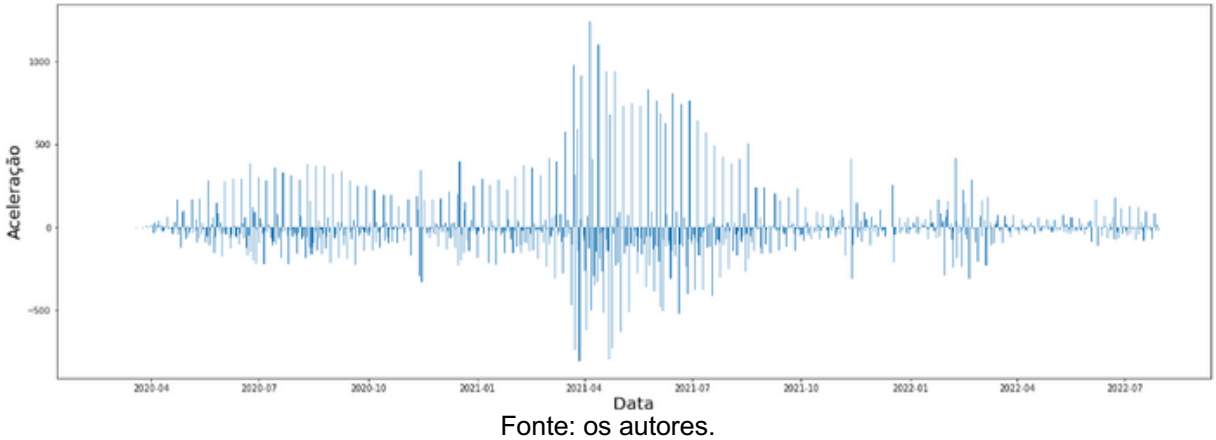
Figura 34 — Função para calcular aceleração de óbitos

```
In [25]: df_data['aceleracaoObitos'] = df_data['obitos_novos'].diff()
```

Fonte: os autores.

Quando ocorre a visualização através do gráfico na Figura 35, notamos que há acelerações e desacelerações e isso demonstra que depois de passar por um período muito crítico entre abril e julho de 2021, a pandemia finalmente foi se enfraquecendo, muito em virtude da vacinação que começou atingir seu ápice nesse mesmo período.

Figura 35 — Gráfico de aceleração de óbitos



Fonte: os autores.

Um dos motivos de alguns dias terem picos altíssimos e outros extremamente baixos é a subnotificação dos casos. Na realidade, os dados sofrem essas distorções principalmente porque os laboratórios e hospitais, durante a pandemia, funcionaram em regime de plantão aos finais de semana, e por isso, muitos desses dados eram represados e divulgados somente no início da semana seguinte. Podemos nomear esse evento como sazonalidade que ocorre quando os dados variam com alguma função temporal. Para suavizar essas distorções utilizamos a média móvel semanal. Com essa média ao longo da semana temos uma suavização. Isso foi possível com a utilização do comando que pode ser visto na Figura 36.

Figura 36 — Comando para calcular a média móvel semanal

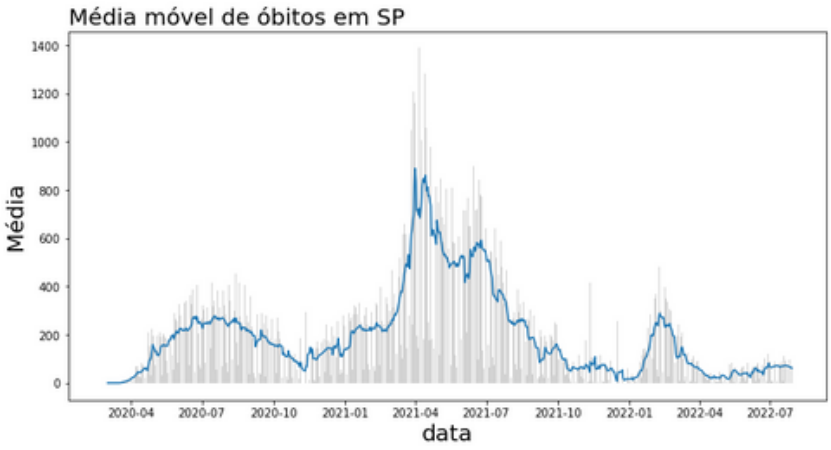
```
In [33]: # Média móvel suaviza a sazonalidade semanal que existe
         # Nesse caso vamos tirar a média da semana
         df_data['mediaObitos'] = df_data.obitos_novos.rolling(window=7, center=False).mean()
         df_data['mediaCasos'] = df_data.casos_novos.rolling(window=7, center=False).mean()
```

Fonte: os autores.

Em seguida, plotamos o gráfico referente a média móvel de casos e óbitos em relação aos casos e óbitos novos (Figura 37).

Figura 37 — Gráfico da média móvel semanal de óbitos

```
In [31]: #Grafico da media móvel de obitos em relação aos Óbitos novos
         plotar_linha('Média móvel de óbitos em SP', 'data', 'Média', 'datahora', 'mediaObitos', df_data, None)
         plt.bar(df_data['datahora'], df_data['obitos_novos'], color='lightgrey')

Out[31]: <BarContainer object of 886 artists>
```

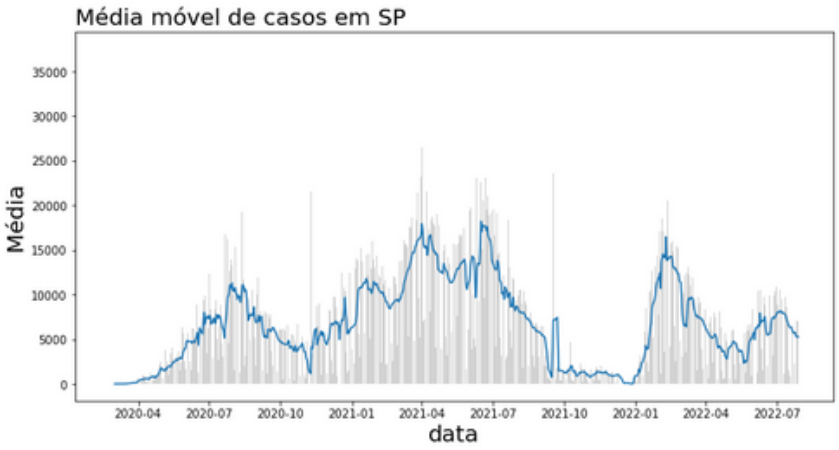


Fonte: os autores.

No gráfico da Figura 38 é possível ver a média móvel semanal dos casos novos.

Figura 38 — Gráfico da média móvel semanal de casos

```
In [32]: #Grafico da media móvel de casos em relação aos casos novos
         plotar_linha('Média móvel de casos em SP', 'data', 'Média', 'datahora', 'mediaCasos', df_data, None)
         plt.bar(df_data['datahora'], df_data['casos_novos'], color='lightgrey')

Out[32]: <BarContainer object of 886 artists>
```



Fonte: os autores.

4 Resultados e Discussões

A análise exploratória realizada nas base de dados da Covid-19 do estado de São Paulo permitiu identificar vários padrões e respostas sobre o impacto da pandemia no mais populoso estado do Brasil. Através dessa análise, identificamos que as cidades com maior quantidade de casos durante o período observado foram a capital São Paulo (mais de 1,1 milhão de casos), Campinas (mais de 185 mil), São José do Rio Preto (mais de 135 mil), São José dos Campos (mais de 124 mil) e Ribeirão Preto (mais de 104 mil). As duas primeiras também aparecem no ranking dos municípios com mais óbitos, assim como Guarulhos (5.454 mortes), São Bernardo do Campo (3.733 mortes) e Santo André (3.458 mortes).

Foi identificado também que a letalidade da doença não está necessariamente ligada a quantidade de casos, visto que as cidades com o maior percentual de mortes em relação ao número de casos registrados foram Sarutaiá (14,8% de letalidade), Sabino (9,9% de letalidade), Salesópolis (8,7% de letalidade), Embaúba (8,5% de letalidade) e Iaras (8% de letalidade), três delas fazem parte do Departamento Regional de Saúde de Bauru, fazendo com que os vereadores do município discutissem o tema em uma Comissão Especial de Inquérito juntamente com a Secretaria de Saúde.

Nessa comissão, segundo o site da Câmara Legislativa de Bauru (2021), o ex-secretário municipal de saúde Sérgio Henrique Antonio foi questionado sobre os motivos que impediram a abertura de leitos clínicos e de UTI no município, frente às altas taxas de ocupação verificadas no segundo semestre de 2020. Segundo informação, Sergio disse que notificou junto aos órgãos competentes a necessidade da abertura de novos leitos. Alguns foram abertos, mas não o suficiente para suprir a demanda da região.

Outra constatação muito clara em relação ao comportamento da pandemia, foi a grande explosão de casos e óbitos no final de 2020 e início de 2021, se estabilizando assim que foi iniciada a campanha de vacinação, quando no dia 17 de janeiro de 2021 foi aplicada a primeira vacina em terras brasileiras. Com o aumento da imunização da população pôde-se perceber a curva de contágio atingindo seu platô.

5 Conclusão

Conclui-se que o presente estudo pode ser aplicado em outras bases de dados, para que novos cenários possam ser traçados e definidos. Utilizando-se dessas informações para que caso surjam novas doenças contagiosas como a Covid-19, os números e padrões levantados podem servir de informação para tomada de decisões de órgãos governamentais e de saúde, bem como políticas públicas para minimizar os seus impactos.

A maior dificuldade encontrada foi manusear a base de dados utilizada no estudo para a inclusão de dados, os quais não se encontravam prontos na base original, como por exemplo, o número de casos acumulados. A familiaridade com a linguagem utilizada no processo de análise de dados exploratória, tornou a pesquisa bem fluida, assim como inúmeros materiais de apoio existentes em relação a ela.

Referências

COELHO, Flavio Codeço. **Computação Científica com Python:** Uma introdução à programação para cientistas. 1 ed. Petrópolis: Lulu.com, v. 1, f. 87, 2007, p. 2-3.

COVID-19. **Ministério da Saúde:** Biblioteca Virtual em Saúde. Disponível em: https://bvsms.saude.gov.br/covid-19-2/. Acesso em: 25 set. 2022.

DADOS ao Cubo: Análise Exploratória de Dados com Python Parte I. *In*: **Dados ao Cubo**: Análise Exploratória de Dados com Python Parte I. [*S. l.*], 27 jul. 2020. Disponível em: https://dadosaocubo.com/analise-exploratoria-de-dados-com-python-parte-i/. Acesso em: 18 ago. 2022.

MCKINNEY, Wes. **Python para análise de dados**: Tratamento de dados com Pandas, NumPy e IPython. 2 ed. São Paulo: Novatec Editora, v. 2, f. 308, 2018, p. 25 e 185-186.

NOGUEIRA, José Vagner Delmiro. CONHECENDO A ORIGEM DO SARS-COV-2 (COVID 19). Revista Saúde e Meio Ambiente, Três Lagoas, v. 11, n. 2, p. 115-124, 9 out 2020.

PYTHON. Python About. Python. 2022. Disponível em: https://www.python.org/about/. Acesso em: 14 ago. 2022.

SCHUCHMANN, Alexandra Zanella *et al*. Isolamento social vertical X Isolamento social horizontal: os dilemas sanitários e sociais no enfrentamento da pandemia de COVID-19. **Brazilian Journal of health Review**, Curitiba, v. 3, n. 3, p. 3556-3576, 30 mar 2020.

SERVIDORES da Secretaria Municipal de Saúde prestam esclarecimentos para a 'CEI da COVID-19'. Câmara Municipal de Bauru. Bauru, 2021. Disponível em: https://www.bauru.sp.leg.br/imprensa/noticias/servidores-da-secretaria-municipal-de-saude-prestam-esclarecimentos-para-a-cei-da-covid-19/. Acesso em: 25 set. 2022.