UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS TRINDADE
CURSO SISTEMAS DE INFORMAÇÃO

PEDRO MATIUCCI PEREIRA

Desenvolvimento de um ETL para Dados de Mortalidade NeoNatal e um Modelo Multivariado de Aprendizado de Máquina para Predição da Taxa de Mortalidade Neonatal

FLORIANÓPOLIS
2025/2

PEDRO MATIUCCI PEREIRA

Desenvolvimento de um ETL para Dados de Mortalidade NeoNatal e um Modelo Multivariado de Aprendizado de Máquina para Predição da Taxa de Mortalidade Neonatal

Trabalho de Conclusão de Curso submetido ao curso de Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Dr. Luís Antonio Lourenço

FLORIANÓPOLIS

2025

RESUMO

A taxa de mortalidade neonatal é um dos principais indicadores utilizados para avaliar a efetividade das políticas públicas e o desempenho da Atenção Primária à Saúde (APS), além de compor o conjunto de indicadores do Plano de Desenvolvimento Sustentável da Organização das Nações Unidas (ONU). No entanto, os dados referentes a esse desfecho costumam levar cerca de um ano para serem consolidados oficialmente, o que gera uma defasagem temporal significativa e dificulta a realização de análises em tempo real. Essa limitação compromete a capacidade de resposta do sistema público de saúde e retarda o desenvolvimento de ações preventivas e políticas baseadas em evidências atualizadas.Diante desse cenário, este trabalho tem como objetivo desenvolver um pipeline de ETL (Extract, Transform and Load) para integrar, transformar e armazenar dados provenientes dos sistemas públicos de informação do DataSUS, especificamente o Sistema de Informações sobre Mortalidade (SIM) e o Sistema de Informações sobre Nascidos Vivos (SINASC), utilizados para o cálculo da taxa de mortalidade neonatal. Além disso, foram incorporados dados do Sistema de Informação em Saúde para a Atenção Básica (SISAB), com o intuito de compor uma abordagem multivariada que considera indicadores relacionados à saúde materno-infantil, como proporção de gestantes com pré-natal adequado, realização de exames e cobertura citopatológica. Com base nesses dados, foi desenvolvido um modelo preditivo multivariado utilizando técnicas de aprendizado de máquina utilizando o algoritmo XGBoost, amplamente reconhecido por sua eficiência em tarefas de regressão. O modelo proposto visa estimar a taxa de mortalidade neonatal de forma antecipada em cidades de grande porte com periodicidade quadrimestral, permitindo análises mais ágeis e contribuindo para o aprimoramento das ações de vigilância e planejamento em saúde pública.O estudo adota a metodologia CRISP-DM e aplica a otimização bayesiana para aprimorar o desempenho do modelo XGBoost. Os resultados obtidos foram comparados com um modelo baseado em séries temporais (SARIMA), demonstrando que o modelo proposto apresentou desempenho superior em todas as métricas avaliadas, incluindo o erro quadrático médio (RMSE), com redução aproximada de 5%, e o erro percentual absoluto médio (MAPE), com diminuição de cerca de 7% no erro.

Palavras-chave: Machine Learning; ETL; Séries temporais; Mortalidade Neonatal.

ABSTRACT

The neonatal mortality rate is one of the main indicators used to assess the effectiveness of public policies and the performance of Primary Health Care (PHC). It is also part of the indicator framework of the United Nations Sustainable Development Goals (SDGs). However, data related to this outcome usually take about a year to be officially consolidated, which creates a significant time lag and hinders real-time analyses. This limitation reduces the responsiveness of the public health system and delays the implementation of preventive actions and evidence-based policies.In this context, this study aims to develop an ETL (Extract, Transform, and Load) pipeline to integrate, transform, and store data from the public information systems managed by DataSUS, specifically the Mortality Information System (SIM) and the Live Birth Information System (SINASC), which are used to calculate the neonatal mortality rate. Additionally, data from the Primary Health Care Information System (SISAB) were incorporated to build a multivariate approach that includes maternal and child health indicators, such as the proportion of pregnant women with adequate prenatal care, the performance of laboratory tests, and cytopathological coverage.Based on these data, a multivariate predictive model was developed using machine learning techniques, particularly the XGBoost algorithm, widely recognized for its efficiency in regression tasks. The proposed model aims to estimate the neonatal mortality rate in large municipalities on a four-month basis, enabling faster analyses and contributing to improvements in health surveillance and planning. The study follows the CRISP-DM methodology and applies Bayesian optimization to enhance model performance. The results were compared with a time series model (SARIMA), demonstrating that the proposed model presented superior results in all evaluated metrics, with an approximately 5% reduction in Root Mean Squared Error (RMSE) and a 7% decrease in Mean Absolute Percentage Error (MAPE).

Keywords: Machine learning; ETL; Time series; Neonatal mortality.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo tem como objetivo apresentar o problema de pesquisa, contextualizar o tema e justificar sua relevância social, científica e tecnológica. Busca-se evidenciar a importância da predição da taxa de mortalidade neonatal no contexto da saúde pública brasileira e o papel das tecnologias de ciência de dados na produção de informações que subsidiem políticas públicas mais ágeis e eficazes.

1.1 CONTEXTUALIZAÇÃO DO TEMA

A taxa de mortalidade neonatal é um indicador-chave para o monitoramento da qualidade da atenção à saúde materno-infantil, além de orientar decisões públicas e alocação de recursos. Este desfecho é utilizado para acompanhar a evolução e a eficácia de políticas públicas, sendo um dos critérios adotados pelo Programa Previne Brasil para a distribuição de receitas entre municípios (MINISTÉRIO DA SAÚDE, 2024).

A relevância do tema também se insere no contexto internacional. A Agenda 2030 da Organização das Nações Unidas (ONU), que tem como meta reduzir a taxa para menos de 12 mortes a cada 1000 nascidos vivos, o que reforça sua relevância global (Garcia et al., 2023). No entanto, apesar dos avanços alcançados, o Brasil ainda apresenta desigualdades regionais significativas e enfrenta desafios relacionados à qualidade e à disponibilidade dos dados oficiais.

As informações sobre mortalidade e natalidade, provenientes de sistemas como o SINASC (Sistema de Informações sobre Nascidos Vivos) e o SIM (Sistema de Informações sobre Mortalidade), podem levar até dois anos para serem consolidadas e disponibilizadas publicamente. Essa defasagem temporal cria uma lacuna crítica que impede o monitoramento em tempo real e limita a capacidade de resposta do sistema público de saúde diante de mudanças rápidas nos indicadores.

Diante desse cenário, torna-se relevante explorar estratégias alternativas e complementares, como o uso de dados disponibilizados no Sistema de Informação em Saúde para a Atenção Básica (SISAB), que reúne indicadores relacionados à atenção pré-natal e ao acompanhamento das gestantes.

Além disso, a combinação entre dados de saúde pública de diferentes fontes e técnicas de análise automatizada tem se mostrado promissora para reduzir

defasagens informacionais e aprimorar a tomada de decisão. No contexto deste estudo, essas técnicas são aplicadas à previsão de séries temporais das taxas de mortalidade neonatal a partir de dados históricos. Modelos de *aprendizado de máquina* destacam-se pela capacidade de capturar padrões complexos nos dados e oferecer predições mais robustas que métodos estatísticos tradicionais, favorecendo análises preditivas mais rápidas e precisas.

1.2 PROBLEMA DE PESQUISA

Apesar da ampla disponibilidade de dados públicos e do avanço de técnicas analíticas, ainda não há uma solução que integre, de forma automatizada e contínua, as etapas de coleta, tratamento e predição dos dados de mortalidade neonatal. Assim, a pergunta de pesquisa que orienta este trabalho pode ser formulado da seguinte forma:

Como desenvolver um processo automatizado de ETL e um modelo de aprendizado de máquina capaz de prever a taxa de mortalidade neonatal nos municípios brasileiros, reduzindo o tempo necessário para análise e tomada de decisão em saúde pública?

1.3 OBJETIVOS

Com o intuito de estabelecer um escopo claro e bem definido para o presente estudo, são apresentados a seguir o objetivo geral da pesquisa e os objetivos específicos.

1.3.1 Objetivo Geral

Desenvolver um processo de ETL para o tratamento e preparação de dados, bem como um modelo de aprendizado de máquina multivariado para a predição da taxa de mortalidade neonatal dos municípios brasileiros.

1.3.2 Objetivos Específicos

Para atingir o objetivo geral proposto, o projeto contempla os seguintes objetivos específicos:

- Identificar as variáveis mais relevantes para a estimativa da mortalidade neonatal;
- Levantar e compreender as bases de dados públicas disponíveis que sejam pertinentes ao problema em questão;
- Desenvolver um pipeline de ETL para extração, transformação e carregamento dos dados relevantes ao cálculo da taxa de mortalidade neonatal;
- Realizar a modelagem e ajuste de hiperparâmetros de um modelo de aprendizado de máquina para a predição da mortalidade neonatal;
- Avaliar o desempenho do modelo preditivo, verificando sua eficácia na estimativa da taxa de mortalidade neonatal.

1.4 JUSTIFICATIVA

A mortalidade neonatal é um dos indicadores mais sensíveis da qualidade da atenção à saúde materno-infantil, além de refletir desigualdades sociais e estruturais do sistema de saúde brasileiro. A demora na consolidação e disponibilização dos dados compromete a capacidade de gestores e pesquisadores em propor intervenções oportunas. Assim, a construção de um modelo preditivo baseado em dados públicos automatizados surge como uma ferramenta potencialmente útil para antecipar tendências, otimizar recursos e subsidiar políticas públicas.Do ponto de vista tecnológico, o trabalho contribui ao integrar técnicas de engenharia de dados (ETL) e modelagem preditiva em um pipeline reproduzível e escalável, reforçando a aplicação prática da ciência de dados no campo da saúde pública.

Do ponto de vista social, os resultados esperados podem apoiar a formulação de estratégias locais e regionais de prevenção e acompanhamento neonatal, promovendo decisões mais rápidas e assertivas no Sistema Único de Saúde (SUS).

2. FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica deste trabalho tem como objetivo fornecer o embasamento necessário para o desenvolvimento de um modelo preditivo da taxa de mortalidade neonatal utilizando técnicas de aprendizado de máquina. Para isso, serão abordados os conceitos fundamentais sobre mortalidade neonatal, discutindo sua relevância como indicador de saúde pública e sua importância para o monitoramento da qualidade da atenção materno-infantil.

Em seguida, serão exploradas as aplicações de aprendizado de máquina em previsões de saúde, com foco nas potencialidades dos algoritmos no contexto de predição de dados de saúde neonatal. Também será realizada uma análise dos processos de ETL (Extract, Transform, Load) aplicados a dados de saúde pública, considerando as especificidades dos sistemas de informação brasileiros.

Por fim, serão discutidos os métodos de validação e avaliação de modelos preditivos em saúde, essenciais para garantir a precisão e a eficácia das predições geradas.

2.1 MORTALIDADE NEONATAL: CONCEITOS E RELEVÂNCIA

A mortalidade neonatal é definida como o número de óbitos de recém-nascidos que ocorrem entre o nascimento e os primeiros 28 dias de vida completos, expressa por 1.000 nascidos vivos. Este indicador é reconhecido internacionalmente como um dos mais sensíveis para avaliar a qualidade da atenção à saúde materno-infantil, refletindo não apenas as condições de nascimento, mas também a eficácia dos cuidados pré-natal, da assistência ao parto e dos cuidados neonatais.

A importância da mortalidade neonatal transcende sua função como indicador sanitário. Ela representa um componente crítico dos Objetivos de Desenvolvimento Sustentável (ODS) da Agenda 2030 da Organização das Nações Unidas, especificamente no ODS 3, que visa "assegurar uma vida saudável e promover o bem-estar para todos, em todas as idades". A meta 3.2 estabelece que os países devem reduzir a mortalidade neonatal para pelo menos 12 óbitos por 1.000 nascidos vivos até 2030 (UNITED NATIONS, 2015).

No contexto brasileiro, a mortalidade neonatal assume particular relevância devido às desigualdades regionais e socioeconômicas existentes no país. Além

disso, o Ministério da Saúde utiliza este indicador como um dos critérios para distribuição de recursos do Programa Previne Brasil, demonstrando sua importância na formulação de políticas públicas e na alocação de recursos para a atenção primária à saúde (MINISTÉRIO DA SAÚDE, 2019).

Apesar da relevância desses dados no contexto brasileiro, as informações sobre mortalidade e natalidade geralmente levam mais de um ano para serem consolidadas e divulgadas oficialmente pelos sistemas SIM e SINASC. A Portaria nº 116/2009 do Ministério da Saúde, que regulamenta o fluxo e a periodicidade desses dados, estabelece que os estados devem enviar as informações em até 60 dias após o mês de ocorrência, com consolidação nacional até 30 de junho do ano seguinte e divulgação definitiva até 30 de dezembro. Esse processo resulta em uma defasagem temporal considerável, que limita a realização de análises em tempo real e dificulta a formulação de políticas públicas baseadas em evidências atualizadas.

2.2 APRENDIZADO DE MÁQUINA EM PREVISÕES DE SAÚDE NEONATAL

O uso de técnicas de inteligência artificial (IA) na área da saúde tem experimentado um crescimento exponencial nas últimas décadas, especialmente no contexto de previsão de desfechos clínicos. Na área da saúde neonatal, essas técnicas têm se mostrado particularmente promissoras devido à complexidade dos fatores que influenciam a mortalidade neonatal e à necessidade de identificar padrões em grandes volumes de dados (MANGOLD et al., 2021).

A aplicação de algoritmos de aprendizado de máquina (ML) na predição de mortalidade neonatal tem sido objeto de crescente interesse na literatura científica. Mangold e colaboradores (2021) conduziram uma revisão sistemática sobre modelos de aprendizado de máquina aplicados à predição da mortalidade neonatal, analisando 45 estudos publicados entre 2000 e 2020. Os resultados demonstraram que modelos de ML como Redes Neurais, *Random Forest* e Árvores de Classificação e Regressão conseguem prever este desfecho de forma precisa, com desempenho superior aos métodos estatísticos tradicionais.

Entre os algoritmos mais utilizados na predição de mortalidade neonatal, destacam-se as árvores de decisão, redes neurais artificiais, máquinas de vetor de suporte (SVM) e algoritmos baseados em boosting, como o XGBoost. O XGBoost, em particular, tem se mostrado eficaz em problemas de classificação e regressão,

apresentando vantagens como capacidade de lidar com dados faltantes, resistência a *overfitting* e um desempenho superior aos modelos comparados (BATISTA et al., 2021).

Batista e colaboradores (2021) desenvolveram um estudo intitulado "Neonatal mortality prediction with routinely collected data: a machine learning approach", no qual compararam cinco modelos de classificação para a predição de mortalidade neonatal: regressão logística, árvores de decisão, *random forest*, XGBoost e redes neurais. O XGBoost apresentou o melhor desempenho, com área sob a curva ROC de 0,85, demonstrando sua eficácia na predição desse desfecho.

Apesar do desempenho superior apresentado por modelos de aprendizado de máquina, sua aplicação prática ainda representa um desafio considerável. Diversos fatores podem comprometer o desenvolvimento de modelos confiáveis e com boa capacidade preditiva. Entre os principais, destaca-se a complexidade na otimização de hiperparâmetros, etapa essencial para equilibrar o modelo e evitar tanto o *overfitting* (excesso de ajuste aos dados de treinamento) quanto o *underfitting* (incapacidade de aprender os padrões relevantes).

Além disso, a qualidade dos dados disponíveis é um ponto crítico. Conjuntos de dados desbalanceados, com alta proporção de valores ausentes ou erros de preenchimento, podem introduzir vieses e comprometer a confiabilidade das previsões. Esses desafios reforçam a necessidade de processos rigorosos de pré-processamento, limpeza e validação dos dados, bem como de técnicas adequadas de regularização e avaliação para garantir resultados consistentes em contextos reais de saúde pública.

2.3 PROCESSOS DE ETL EM DADOS DE SAÚDE PÚBLICA

O processo de ETL (*Extract, Transform, Load*) constitui um componente fundamental em projetos de ciência de dados. Este processo envolve a extração de dados de diversas fontes, sua transformação para um formato adequado à análise e o carregamento em um sistema de armazenamento estruturado (KIMBALL; CASERTA, 2004).

No contexto brasileiro, os sistemas de informação em saúde apresentam características específicas que tornam os processos de ETL particularmente desafiadores. O Sistema de Informações sobre Nascidos Vivos (SINASC), o Sistema

de Informações sobre Mortalidade (SIM) e o Sistema de Informação em Saúde para a Atenção Básica (SISAB) são as principais fontes de dados para estudos de mortalidade neonatal, cada um com suas particularidades estruturais e de disponibilização (MINISTÉRIO DA SAÚDE, 2011).

O SINASC, implantado em 1990, coleta dados sobre nascimentos através da Declaração de Nascido Vivo (DN), fornecendo informações sobre características maternas, do parto e do recém-nascido. O SIM, por sua vez, coleta dados sobre óbitos através da Declaração de Óbito (DO), incluindo informações sobre as causas de morte e características do falecido. O SISAB, mais recente, disponibiliza indicadores de saúde da atenção primária, incluindo dados sobre cobertura de pré-natal e outros indicadores relevantes para a predição de mortalidade neonatal (CESAR, 2024).

A experiência documentada por Cesar (2024) no desenvolvimento de um modelo preditivo dos anos de vida perdidos por morte prematura demonstra a viabilidade de implementar processos de ETL eficientes para dados de saúde brasileiros. O autor utilizou Python como linguagem de programação, pandas para manipulação de dados e Google BigQuery como plataforma de armazenamento, criando um pipeline robusto e escalável.

Esses processos compõem a base estrutural do pipeline de dados proposto neste trabalho, assegurando a integridade, consistência e reprodutibilidade das informações utilizadas no desenvolvimento e avaliação do modelo preditivo, viabilizando a automação da coleta e atualização contínua dos indicadores de saúde pública.

2.4 ANÁLISE DE SÉRIES TEMPORAIS

Séries temporais são coleções de observações registradas em instantes ou intervalos regulares (por exemplo, dias, meses ou quadrimestres), em que a ordem temporal e a dependência entre observações são elementos estruturantes do processo gerador dos dados (MONTGOMERY; PECK; VINING, 2015). A análise temporal busca identificar e quantificar componentes como tendência (movimento de longo prazo), sazonalidade (padrões periódicos), ciclicidade (oscilações não

estritamente periódicas) e ruído, além de detectar rupturas e mudanças de regime (SHUMWAY; STOFFER, 2017).

No campo da saúde pública, essa abordagem é fundamental para o monitoramento e a previsão de indicadores, permitindo avaliar o impacto de eventos e políticas ao longo do tempo e apoiar decisões tempestivas (BHASKARAN et al., 2013). Para a mortalidade neonatal, a modelagem por séries temporais possibilita antecipar flutuações, comparar períodos e investigar efeitos de determinantes externos, como epidemias, intervenções e alterações demográficas.

O processo de modelagem usualmente inicia com análise exploratória (visualização, autocorrelação, estacionariedade, identificação de sazonalidade e pontos de ruptura), prossegue com a especificação do modelo e culmina em avaliação preditiva (BOX; JENKINS; REINSEL, 2015). Entre os métodos clássicos destacam-se:

- ARIMA, adequado para séries com estrutura autoregressiva e de médias móveis, após eventuais diferenciações para remover tendência;

- SARIMA, que estende o ARIMA com termos sazonais, aumentando a capacidade de capturar periodicidades típicas de indicadores de saúde (CHECHI; BAYER, 2012).

Em paralelo aos modelos tradicionais, técnicas de aprendizado de máquina têm sido empregadas quando se deseja capturar não linearidades e interações complexas. O algoritmo XGBoost, baseado em árvores de decisão com gradiente reforçado, é frequentemente utilizado para predição. Apesar de seu bom desempenho em cenários tabulares, sua aplicação em séries temporais requer validação temporal adequada e atenção ao overfitting, dada a alta flexibilidade do modelo (PALIARI; KARANIKOLA; KOTSIANTIS, 2021).

Assim, a escolha entre modelos clássicos de séries temporais (ARIMA/SARIMA) e modelos de aprendizado de máquina depende da estrutura dos dados, da presença de sazonalidade marcada, da disponibilidade de variáveis explicativas e do objetivo do estudo. Em todos os casos, uma estratégia de

validação que respeite a ordem temporal é indispensável para obter estimativas confiáveis de desempenho.

Dessa forma, o modelo proposto neste trabalho combina técnicas de aprendizado de máquina com princípios de modelagem de séries temporais para antecipar variações quadrimestrais na mortalidade neonatal.

2.5 OTIMIZAÇÃO DE HIPERPARÂMETROS

A otimização de hiperparâmetros é uma etapa crucial no desenvolvimento de modelos de aprendizado de máquina, influenciando diretamente o desempenho e a capacidade de generalização do modelo. No contexto de modelos preditivos para saúde, a escolha adequada dos hiperparâmetros pode significar a diferença entre um modelo clinicamente útil e um modelo inadequado para aplicação prática (BERGSTRA; BENGIO, 2012).

Tradicionalmente, a otimização de hiperparâmetros é realizada através de métodos como *Grid Search* ou *Random Search*. O *Grid Search* explora exaustivamente um espaço de hiperparâmetros predefinido, enquanto o *Random Search* seleciona aleatoriamente combinações de hiperparâmetros para avaliação. Embora eficazes, esses métodos podem ser computacionalmente caros, especialmente quando o espaço de hiperparâmetros é grande (BERGSTRA; BENGIO, 2012).

A Otimização Bayesiana, por outro lado, emergiu como uma alternativa mais eficiente, utilizando modelos probabilísticos para guiar a busca pelos hiperparâmetros ótimos. A superioridade da otimização bayesiana foi demonstrada de forma contundente na *Black-Box Optimization Challenge* 2020, onde os melhores participantes apresentaram eficiência até 100 vezes maior do que a obtida com busca aleatória. Todos os 20 melhores colocados utilizaram métodos baseados em modelos substitutos, como a otimização bayesiana, reforçando sua eficácia em tarefas de ajuste de hiperparâmetros (TURNER et al., 2021).

Mesmo com o uso de algoritmos de otimização avançados, é fundamental evitar o sobreajuste (*overfitting*), situação em que o modelo “memoriza” os dados de treinamento, perdendo sua capacidade de generalização para novos conjuntos de

dados. Para mitigar esse problema, além da escolha adequada dos hiperparâmetros, é essencial adotar estratégias de validação, como a *cross-validation*, que permitem avaliar o desempenho do modelo em diferentes subconjuntos e garantir que ele mantenha uma performance consistente também fora da amostra de treinamento.

2.6 FERRAMENTAS E PLATAFORMAS PARA ANÁLISE DE DADOS DE SAÚDE

O desenvolvimento de modelos preditivos para dados de saúde requer o uso de ferramentas e plataformas adequadas que possam lidar com as especificidades desses dados, incluindo volume, variedade e necessidade de segurança. A escolha das ferramentas adequadas pode influenciar significativamente a eficiência do desenvolvimento e a qualidade dos resultados obtidos (RAJKOMAR; DEAN; KOHANE, 2018).

A linguagem Python tem se consolidado como uma das principais escolhas para projetos de ciência de dados na área da saúde, devido à sua ampla gama de bibliotecas especializadas e à facilidade de integração com diferentes sistemas. Bibliotecas como pandas, scikit-learn, statsmodels e matplotlib fornecem funcionalidades essenciais para manipulação de dados, desenvolvimento de modelos e visualização de resultados (MCKINNEY, 2010).

O Google BigQuery tem emergido como uma plataforma robusta para armazenamento e análise de grandes volumes de dados de saúde. Sua capacidade de processar consultas SQL em petabytes de dados, combinada com recursos de aprendizado de máquina integrados, torna-a particularmente adequada para projetos que envolvem dados de saúde pública em larga escala (GOOGLE CLOUD, 2021).

A integração entre diferentes ferramentas e plataformas é fundamental para criar pipelines eficientes de análise de dados de saúde. A experiência documentada por Cesar (2024) demonstra como a combinação de Python para processamento de dados, Google BigQuery para armazenamento e bibliotecas especializadas para machine learning pode resultar em um ambiente de desenvolvimento robusto e escalável.

Essas ferramentas dão suporte tanto à etapa de ETL, especialmente no armazenamento e organização dos dados, garantindo que eles sejam facilmente

acessíveis e consistentes, quanto ao treinamento do modelo preditivo, possibilitando que os dados sejam utilizados sem a necessidade de transformações substanciais adicionais. Além disso, essas plataformas favorecem a reprodutibilidade do trabalho, permitindo que o mesmo conjunto de dados seja processado e analisado em diferentes ambientes de desenvolvimento de forma padronizada.

2.7 AVALIAÇÃO DE MODELOS PREDITIVOS

Para problemas de regressão, como a predição de taxas de mortalidade neonatal, métricas como o Erro Quadrático Médio (MSE), a Raiz do Erro Quadrático Médio (RMSE) e o Erro Percentual Absoluto Médio (MAPE) são amplamente utilizadas. O MSE e o RMSE medem a magnitude média dos erros ao penalizar de forma mais intensa desvios maiores, o que os torna adequados para identificar modelos com erros extremos. O RMSE, em particular, é amplamente empregado por fornecer o erro na mesma unidade da variável predita, o que facilita sua interpretação. Já o coeficiente de determinação (R^2) quantifica a proporção da variabilidade dos dados explicada pelo modelo. Valores elevados de R^2 indicam maior capacidade preditiva global, enquanto valores reduzidos sinalizam que parte substancial da variabilidade permanece não explicada.

O MAPE é especialmente útil em contextos de saúde pública, pois expressa o erro em termos percentuais, tornando-se intuitivo para profissionais e gestores. Por exemplo, um MAPE de 0,05 indica que as previsões do modelo apresentam, em média, erro de 5 por cento em relação aos valores reais. Entretanto, o MAPE apresenta limitações conhecidas, sendo sensível a valores reais próximos de zero e tendendo a favorecer previsões subestimadas. Por esse motivo, recomenda-se que ele seja interpretado em conjunto com o RMSE e com o coeficiente de determinação (R^2), oferecendo uma avaliação mais abrangente do desempenho do modelo (HYNDMAN; KOEHLER, 2006).

A validação cruzada temporal é particularmente relevante para modelos que utilizam dados de séries temporais, como é o caso da predição de mortalidade neonatal. Esta técnica garante que o modelo seja testado em períodos posteriores aos utilizados para treinamento, simulando condições reais de uso onde as predições são feitas para períodos futuros (TASHMAN, 2000).

A comparação com modelos baseline, incluindo métodos estatísticos tradicionais, como ARIMA e SARIMA, é essencial para demonstrar a superioridade no ganho de informação das técnicas de aprendizado de máquina. Esta comparação fornece evidências quantitativas dos benefícios da abordagem proposta e justifica a complexidade adicional dos modelos (HYNDMAN; ATHANASOPOULOS, 2018).

Além das métricas de desempenho, a literatura enfatiza a importância de boas práticas de reprodutibilidade, como a separação temporal adequada entre os conjuntos de treino e teste, o uso de *random seeds* (sementes aleatórias fixas) e a documentação das versões das bibliotecas e scripts utilizados. Tais práticas garantem a rastreabilidade e a replicabilidade dos resultados obtidos.

Em resumo, a fundamentação teórica apresentada demonstra que o desenvolvimento de modelos preditivos para mortalidade neonatal utilizando técnicas de inteligência artificial é uma abordagem promissora e bem fundamentada na literatura científica. A combinação de algoritmos eficazes, processos de ETL robustos e técnicas adequadas de avaliação fornece a base necessária para o desenvolvimento de uma modelagem precisa.

3 TRABALHOS RELACIONADOS

O objetivo desta seção é apresentar e discutir estudos que abordam a predição de mortalidade neonatal/infantil com técnicas de aprendizado de máquina, a fim de contextualizar a literatura e evidenciar as contribuições do presente trabalho. Para esta revisão focal, consideraram-se três referências recentes e diretamente pertinentes: (i) um estudo aplicado com dados rotineiros de nascimentos e múltiplos algoritmos de machine learning (Batista et al., 2021), (ii) uma revisão sistemática de modelos de ML para mortalidade neonatal (Mangold et al., 2021), e (iii) uma tese de doutorado que desenvolve modelos preditivos baseados em inteligência artificial para mortalidade infantil no estado do Ceará (Aguiar, 2019).

3.1 TRABALHOS ANALISADOS

O estudo de Batista e colaboradores (2021) representa uma das pesquisas mais relevantes e recentes no contexto brasileiro sobre a utilização de aprendizado de máquina para prever a mortalidade neonatal a partir de dados de rotina dos sistemas públicos de saúde. O trabalho utilizou uma base de dados abrangente, composta por 1.130.000 nascimentos e 7.282 óbitos neonatais registrados na cidade de São Paulo entre 2012 e 2017, integrando informações provenientes dos sistemas SINASC (Sistema de Informações sobre Nascidos Vivos) e SIM (Sistema de Informação de Mortalidade).

Os autores compararam cinco modelos de aprendizado de máquina para classificação: Regressão Logística, Árvore de Decisão, Random Forest, XGBoost e Rede Neural Artificial, utilizando métricas como AUC, sensibilidade, especificidade e F1-Score para avaliar o desempenho. O modelo XGBoost obteve o melhor resultado, com AUC = 0,971, o que significa que ele foi capaz de identificar corretamente 97,1% dos casos de óbito neonatal.

Um dos achados mais interessantes do estudo é que o modelo manteve desempenho satisfatório mesmo quando treinado com um conjunto reduzido de variáveis, o que demonstra a possibilidade de implementação prática em contextos com limitação de dados. Entre as variáveis de maior importância para o modelo, destacam-se o peso ao nascer, a idade gestacional, o Apgar no primeiro minuto (Uma escala clínica que avalia sinais fisiológicos do recém nascido), o tipo de parto e a idade materna.

O trabalho conclui que os modelos de aprendizado de máquina podem auxiliar gestores e profissionais da saúde na identificação precoce de riscos neonatais, permitindo intervenções preventivas mais direcionadas. Entretanto, os autores reconhecem limitações, como a ausência de validação externa (em outros estados ou períodos) e o fato de os dados serem restritos a uma única região urbana, o que pode limitar a generalização dos resultados.

O estudo de Mangold e colaboradores (2021) consiste em uma revisão sistemática da literatura internacional sobre o uso de modelos de aprendizado de máquina para a predição de mortalidade neonatal. O objetivo dos autores foi sintetizar e avaliar criticamente os métodos, variáveis e resultados reportados em estudos realizados entre 2000 e 2020.

A revisão identificou 11 estudos relevantes, abrangendo cerca de 1,26 milhão de neonatos e diferentes contextos geográficos. Foram analisados algoritmos como árvores de decisão, máquinas de vetor de suporte (SVM), redes neurais artificiais, *random forest* e XGBoost, comparados a modelos estatísticos tradicionais, como regressão logística. Os valores de AUC variaram de 0,58 a 0,97, demonstrando grande heterogeneidade entre os resultados e os contextos de aplicação.

Entre as variáveis preditoras mais recorrentes estão o peso ao nascer, a idade gestacional, a escala de Apgar, o sexo do recém-nascido, o tipo de parto e a paridade materna. Contudo, os autores observaram que poucos estudos relataram informações sobre calibração dos modelos, validação externa e interpretação clínica dos resultados, o que limita a adoção dessas soluções em ambientes de saúde reais.

A revisão conclui que, embora as técnicas de aprendizado de máquina apresentem elevado potencial preditivo, a área ainda carece de padronização metodológica, transparência nos relatórios e validação multicêntrica, de modo a garantir a reprodutibilidade e o uso responsável desses modelos em políticas de saúde.

A tese de doutorado de Aguiar (2019) é uma das poucas produções acadêmicas brasileiras que exploram a mortalidade infantil sob a ótica da inteligência artificial em um contexto regional. O autor desenvolveu um modelo preditivo para identificar fatores de risco e prever o óbito infantil (até 1 ano de vida)

utilizando dados de 2013 a 2017 do estado do Ceará, totalizando 8.159 óbitos e 10.235 nascidos vivos sobreviventes.

A metodologia empregada baseou-se em redes neurais artificiais (MLP), com validação cruzada e diferentes combinações de variáveis socioeconômicas, demográficas e assistenciais. O modelo alcançou acurácia média de 97%, mostrando alto poder discriminativo entre óbitos e sobreviventes. As variáveis mais relevantes para o modelo foram o peso ao nascer, a idade gestacional, o número de consultas pré-natais, a idade materna e o nível de escolaridade da mãe.

Além da modelagem preditiva, o estudo também teve uma dimensão aplicada, desenvolvendo mapas de risco para auxiliar a priorização de municípios com maiores probabilidades de mortalidade infantil. O autor argumenta que o uso de técnicas de IA pode se tornar uma ferramenta essencial para o planejamento e avaliação das políticas públicas de saúde, especialmente em estados com desigualdades regionais acentuadas.

Entre as limitações apontadas, destaca-se a restrição geográfica da amostra, que se concentra em um único estado, e a ausência de integração com bases nacionais de dados (como SISAB ou SINASC completo), o que reduz a generalização dos resultados para o território brasileiro como um todo.

3.2 COMPARAÇÃO CRÍTICA E LACUNAS DA LITERATURA

A análise dos trabalhos relacionados evidencia diferenças significativas quanto à unidade de análise, à natureza dos dados utilizados e ao tipo de modelagem proposta. Os estudos de Batista et al. (2021) e Aguiar (2019) concentram-se na classificação individual do risco de óbito neonatal ou infantil, utilizando dados de características maternas e do recém-nascido. Em contraste, o presente trabalho propõe uma abordagem voltada a séries temporais agregadas por município, com horizonte quadrimestral, direcionada ao monitoramento e planejamento em saúde pública e não à triagem clínica de casos individuais.

No que diz respeito à validação e generalização dos modelos, observa-se uma limitação comum nos estudos revisados. A revisão sistemática de Mangold et al. (2021) destaca a escassez de validação externa e a ausência de calibração adequada em grande parte dos modelos avaliados, o que compromete a

reprodutibilidade e a aplicação prática dos resultados. O trabalho de Batista et al. (2021) apresenta excelente desempenho preditivo, porém restrito a um único cenário urbano (São Paulo), enquanto a tese de Aguair(2019) tem abrangência regional, concentrando-se no estado do Ceará. Portanto, ainda não há evidências consolidadas sobre o desempenho de modelos de aprendizado de máquina em múltiplos contextos brasileiros, especialmente em municípios de grande, médio e pequeno porte.

Quanto às variáveis e fontes de dados, nota-se uma predominância de preditores clássicos como peso ao nascer, idade gestacional, e idade materna nos estudos de Batista et al. (2021) e Mangold et al. (2021). O presente trabalho, inova ao empregar indicadores assistenciais da Atenção Primária à Saúde (APS), extraídos do SISAB, como variáveis exógenas multivariadas. Entre elas, destacam-se a proporção de gestantes com seis ou mais consultas pré-natais, com exames realizados e com atendimento odontológico. Essa abordagem amplia o escopo da literatura existente ao relacionar desempenho dos serviços de saúde e resultados neonatais, algo ainda pouco explorado em estudos anteriores.

Outra diferença fundamental está na tarefa preditiva. Enquanto a maior parte da literatura trata o problema como uma classificação binária (óbito/sobrevivência), esse trabalho o formula como uma regressão de séries temporais buscando prever valores contínuos de taxa de mortalidade neonatal por município. Para isso, utiliza métricas de erro específicas para regressão (MAPE, RMSE e R^2) e aplica validação cruzada temporal do tipo *expanding window*, garantindo que o modelo fosse avaliado apenas em períodos posteriores aos utilizados no treinamento, preservando a ordem cronológica dos dados, comparando o desempenho de algoritmos de aprendizado de máquina (como o XGBoost) com modelos estatísticos clássicos (ARIMA, SARIMA). Essa abordagem permite avaliar não apenas a precisão, mas também a estabilidade do modelo ao longo do tempo.

Em termos de pipeline e reprodutibilidade, a revisão sistemática evidencia uma grande heterogeneidade metodológica e falta de padronização entre os estudos analisados (MANGOLD et al., 2021). O trabalho de Batista et al. (2021) apresenta boa documentação dos experimentos, mas não descreve um processo de ETL automatizado e multi-fonte. Já a tese de Aguiar(2019) demonstra competência técnica na modelagem, mas não inclui uma camada de orquestração ou versionamento em nuvem. Assim, persiste ainda na literatura uma lacuna

relacionada à automação e escalabilidade dos pipelines de dados, aspectos essenciais para a aplicação prática e recorrente dessas soluções no monitoramento em saúde pública.

Por fim, destaca-se a defasagem informacional como um desafio recorrente não abordado pelos estudos analisados. As bases oficiais de mortalidade e natalidade (SIM e SINASC) frequentemente apresentam um atraso superior a um ano até sua consolidação, o que limita a capacidade de análise em tempo quase real. Nesse sentido, há uma oportunidade de avanço metodológico com a utilização de dados mais atualizados do SISAB, permitindo a aplicação de técnicas de *nowcasting* e *forecasting* para reduzir o hiato temporal entre a coleta e a disponibilidade das informações.

Em síntese, a revisão dos trabalhos demonstra que, embora existam estudos relevantes voltados à classificação de risco e à análise preditiva da mortalidade neonatal, ainda são escassas as pesquisas que integram múltiplas fontes de dados públicas, incorporam variáveis assistenciais e utilizam abordagens temporais agregadas e automatizadas. Essas lacunas justificam e fundamentam o desenvolvimento do modelo proposto neste trabalho.

3.3 DIFERENCIAIS E CONTRIBUIÇÕES DO PRESENTE ESTUDO

O presente trabalho propõe a predição multivariada da taxa municipal de mortalidade neonatal, baseada em séries temporais quadrimestrais, alinhando-se às necessidades de gestão territorial e vigilância em saúde. Outro diferencial importante é o uso de variáveis exógenas da Atenção Primária à Saúde (SISAB), que introduz indicadores assistenciais como preditores antecedentes, permitindo antecipar variações na taxa de mortalidade a partir da qualidade da atenção pré-natal.

No que se refere à avaliação do modelo, é implementada uma validação cruzada temporal, com comparação direta entre o modelo SARIMA, reportando métricas como MAPE, RMSE e R^2 e discutindo suas limitações conforme boas práticas da literatura. Além disso, o estudo desenvolve um pipeline de ETL automatizado e reprodutível, responsável pela extração, transformação e carregamento de dados integrados entre os sistemas SINASC, SIM e SISAB, hospedados no Google BigQuery.

Outro aspecto relevante é a mitigação da defasagem informacional por meio de uma arquitetura voltada a *nowcasting* e *forecasting*, reduzindo o intervalo entre o registro e a análise dos dados e possibilitando decisões mais oportunas na Atenção Primária à Saúde.

Em contraste com a ênfase predominante em modelos individuais e retrospectivos, este trabalho avança o estado da arte ao unir modelagem temporal multivariada, dados assistenciais atualizados, validação comparativa e automação de pipeline, resultando em uma ferramenta de alto valor prático para o monitoramento contínuo e a formulação de políticas públicas em saúde no Brasil.

4 METODOLOGIA

De acordo com o escopo do projeto, foram adotadas duas metodologias complementares: uma voltada ao processo de ETL (Extract, Transform and Load) e outra ao desenvolvimento do modelo de aprendizado de máquina (ML).

No processo de ETL, foi desenvolvido um script em Python responsável por realizar o download automático dos dados de natalidade e mortalidade, disponibilizados pelos sistemas SINASC e SIM, por meio de requisições HTTP (requests) a URLs públicas. Essa implementação foi baseada no trabalho de Cesar (2024), que desenvolveu um ETL com estrutura semelhante para dados de mortalidade, com o objetivo de calcular o indicador anos de vida perdidos por morte prematura (YLL, *Years of Life Lost*). Como as bases de mortalidade utilizadas são as mesmas e a de natalidade apresenta estrutura similar, esse trabalho serviu como referência para a construção do ETL proposto.

Para isso, foi realizado um fork do repositório público disponibilizado pelo autor, disponível em: https://codigos.ufsc.br/marco.cesar/yll_time_series_machine_learning. A partir dessa base, foram realizadas adaptações e melhorias necessárias para atender aos objetivos e ao escopo deste estudo.

Entretanto, diferentemente do SIM e do SINASC, os dados do SISAB não estão disponíveis por meio de APIs ou arquivos tabulados em repositórios padronizados. Por essa razão, foi desenvolvido um *web scraper* personalizado, capaz de extrair as informações diretamente do portal público do sistema, respeitando as limitações técnicas e éticas envolvidas nesse tipo de coleta automatizada.

Posteriormente, os dados foram tratados com o auxílio da biblioteca Pandas, garantindo a padronização e limpeza das tabelas. As bases resultantes foram armazenadas em nuvem, no Google BigQuery, de modo a facilitar o treinamento dos modelos e as análises subsequentes, o código desenvolvido nessa etapa está disponível no repositório códigos Ufsc (https://codigos.ufsc.br/tcc_etl_modelo_multivariado_previsao_mortalidade/etl_tcc/-/tree/763a67c1321ddaf6c8fbd27d044356c3c6604185/) e na seção de apêndices.

No desenvolvimento do modelo de aprendizado de máquina foi adotada uma segmentação da metodologia CRISP-DM (*Cross-Industry Standard Process for Data Mining*) utilizando elementos dessa metodologia, amplamente utilizada em projetos de ciência de dados. Essa abordagem estrutura o processo em seis etapas principais: (i) compreensão do negócio, (ii) compreensão dos dados, (iii) preparação dos dados, (iv) modelagem, (v) avaliação e (vi) implantação.

4.1 COMPREENSÃO DO NEGÓCIO

Nesta fase, busca-se compreender o domínio do problema e definir claramente os objetivos do projeto. Foram analisadas a forma de armazenamento dos dados, as informações essenciais para o desenvolvimento do modelo preditivo e a adequação das ferramentas escolhidas, como o Google BigQuery e a linguagem Python, aos propósitos do trabalho.

Durante essa etapa, foram identificadas as URLs utilizadas para o download dos dados do ETL, bem como os dicionários de dados que descrevem o significado de cada variável presente nas bases. Também foi avaliada a forma como os dados são disponibilizados nos sistemas públicos, observando-se que as informações do SISAB estão organizadas em frequência quadrimestral e abrangem o período de 2022 a 2025.

Com base nessa estrutura, os dados de mortalidade (SIM) e natalidade (SINASC) foram convertidos para a mesma frequência quadrimestral, permitindo a integração entre as diferentes fontes. Nessa fase também foi revisada a fórmula de cálculo da taxa de mortalidade neonatal, definida como o número de óbitos de recém-nascidos com até 28 dias de vida dividido pelo número de nascidos vivos e multiplicado por mil.

4.2 COMPREENSÃO DOS DADOS

Com os dados e o escopo do trabalho bem definidos, iniciou-se a etapa de compreensão dos dados, que teve como objetivo conhecer em profundidade as bases utilizadas e avaliar sua qualidade. As informações foram obtidas a partir dos sistemas SIM, SINASC e SISAB, que juntos formam a base principal para o cálculo da taxa de mortalidade neonatal.

Nesta etapa também foi escolhido os seguintes indicadores do SISAB, devido a sua correlação direta com a mortalidade neonatal:

- Proporção de gestantes com pelo menos seis consultas de pré-natal realizadas, sendo a primeira até a 12ª semana de gestação;
- Proporção de gestantes com realização de exames para sífilis e HIV;
- Proporção de gestantes com atendimento odontológico realizado;
- Proporção de mulheres com coleta de exame citopatológico na Atenção Primária à Saúde

Os arquivos foram carregados e analisados em ambiente Python, utilizando principalmente a biblioteca Pandas para leitura e manipulação das tabelas. Foram também empregadas bibliotecas de visualização, como Matplotlib e Seaborn, para facilitar a análise gráfica das séries e das distribuições.

Foi realizada uma análise exploratória dos dados (EDA) que buscou identificar padrões sazonais, a evolução da taxa de mortalidade neonatal ao longo do tempo e a distribuição temporal das variáveis. Além disso, foi verificada a consistência entre os dados provenientes das diferentes fontes e a correspondência entre o número de nascidos vivos e o número de óbitos registrados.

Durante essa fase, também foram calculadas estatísticas descritivas, como médias, desvios padrão e proporção de valores nulos, de modo a compreender melhor a estrutura e a completude das informações. Observou-se que alguns municípios apresentavam variações consideráveis na qualidade dos registros, o que foi considerado nas etapas seguintes de preparação e filtragem dos dados.

4.3 PREPARAÇÃO DOS DADOS

Nessa etapa, os dados consolidados no Google BigQuery foram extraídos e processados em ambiente Python, utilizando um notebook dedicado à preparação para o modelo de aprendizado de máquina. As transformações realizadas foram relativamente simples, uma vez que grande parte do pré-processamento já havia sido executada no pipeline de ETL.

Inicialmente, as tabelas foram agrupadas e ajustadas para garantir a granularidade quadrimestral, compatível com a frequência dos indicadores do SISAB. Em seguida, aplicou-se um filtro para selecionar apenas municípios de grande porte, definidos como aqueles com população entre 100 mil e 900 mil habitantes, conforme a classificação do IBGE.

Essa separação dos dados de municípios de grande porte foi realizada porque localidades com menos de 100 mil habitantes apresentaram um alto volume de registros nulos e inconsistências, como períodos com número de óbitos superior ao de nascidos vivos, o que indica possíveis erros no preenchimento das declarações.

Dessa forma, optou-se por excluir esses municípios na etapa de modelagem, buscando melhorar o desempenho e a confiabilidade do modelo preditivo. No entanto, o pipeline de ETL desenvolvido mantém o processamento completo de todos os municípios, sendo o filtro aplicado apenas na fase de preparação dos dados para o treinamento do modelo.

Ademais foram removidos valores atípicos pelo método do intervalo interquartil (IQR) e registros inconsistentes, como municípios que apresentavam número de óbitos superior ao de nascidos vivos.

Além disso, foram excluídos todos os municípios cuja taxa de mortalidade apresentava mais de 80% de valores ausentes, garantindo a consistência da variável-alvo. A taxa de mortalidade neonatal foi então recalculada e os dados foram agregados por município, ano e quadrimestre, totalizando 1.751 registros e 9 variáveis no intervalo de 2022 ao primeiro quadrimestre de 2025.

O quadrimestre foi transformado em uma variável trigonométrica, utilizando funções seno e cosseno, de modo a representar a sazonalidade cíclica do tempo. As demais variáveis utilizadas como preditoras foram os indicadores assistenciais do SISAB, incluindo a proporção de gestantes com seis consultas pré-natal, realização de exames de sífilis e HIV, atendimento odontológico, coleta de citopatológico, além da população e da unidade federativa.

Para o treinamento do modelo, os dados foram divididos respeitando a ordem temporal: as observações até o primeiro quadrimestre de 2024 foram utilizadas para treino e os dois quadrimestres finais de 2024 foram reservados para teste. Não foi aplicada validação cruzada adicional, pois o número de períodos

disponíveis era reduzido e a divisão temporal já assegurava independência entre os conjuntos.

Todo o processamento foi realizado em Python, com apoio das bibliotecas pandas, numpy, matplotlib, seaborn e scikit-learn para manipulação e visualização dos dados, além de xgboost e bayes_opt para a etapa de modelagem e otimização. Durante a busca pelos melhores hiperparâmetros, foi utilizado o método *BayesSearchCV* em conjunto com a técnica de validação temporal *TimeSeriesSplit*, garantindo que a avaliação respeitasse a sequência cronológica dos dados e evitasse vazamento de informação entre treino e teste. O conjunto final de dados apresentou estrutura limpa, padronizada e sem inconsistências, estando pronto para o treinamento do modelo preditivo XGBoost.

4.4 MODELAGEM

Com o conjunto de dados preparado, iniciou-se a etapa de modelagem, na qual foram testados diferentes algoritmos de predição com o objetivo de identificar aquele com melhor desempenho para a estimativa da taxa de mortalidade neonatal. Foram avaliados os modelos de Regressão Linear Múltipla, SARIMAX e XGBoost, todos aplicados sobre a mesma base de dados e avaliados de forma comparável.

As métricas utilizadas para a análise de desempenho foram o Erro Quadrático Médio (MSE), a Raiz do Erro Quadrático Médio (RMSE), o Coeficiente de Determinação (R^2) e o Erro Percentual Absoluto Médio (MAPE).

O XGBoost foi escolhido por seu desempenho consistente em tarefas de regressão tabular e pela capacidade de lidar com relações não lineares entre as variáveis. Para otimizar seu desempenho, foi utilizada a técnica de Otimização Bayesiana, conduzida por meio do método *BayesSearchCV*, que busca os melhores hiperparâmetros de forma probabilística e iterativa, com eficiência amostral superior ao *Random Search* ou *Grid Search* (TURNER et al., 2021).

O espaço de busca contemplou os principais hiperparâmetros do modelo, incluindo taxa de aprendizado (learning_rate), profundidade máxima (*max_depth*), número de estimadores (*n_estimators*), pesos mínimos das folhas (*min_child_weight*), taxa de amostragem (*subsample*), proporção de colunas (*colsample_bytree* e *colsample_bylevel*), regularizações (*reg_alpha* e *reg_lambda*), número máximo de bins (*max_bin*) e parâmetro de complexidade (*gamma*). Foram

realizadas 100 iterações de busca bayesiana, utilizando a métrica de R^2 como função objetivo.

Tabela 1 – Espaço de Busca Otimização Bayesiana

Hiperparâmetro	Mínimo	Máximo
`learning_rate`	0.001	0.3
`n_estimators`	100	5000
`max_depth`	2	30
`min_child_weight`	1	50
`subsample`	0.5	1.0
`colsample_bytree`	0.5	1.0
`reg_alpha`	0.0	10.0
`reg_lambda`	0.001	100.0
`gamma`	0.0	10.0
`max_bin`	32	512
`colsample_bylevel`	0.3	1.0

Fonte: elaborado pelo autor.

Para garantir uma avaliação temporal adequada e evitar vazamento de informações entre períodos, foi aplicada validação cruzada específica para séries temporais, por meio do método *TimeSeriesSplit* com dois subconjuntos (*folds*), maximizando o tamanho do conjunto de treino em cada iteração.

4.5 AVALIAÇÃO

A avaliação do modelo foi conduzida utilizando o conjunto de teste correspondente aos dois últimos quadrimestres de 2024, enquanto os dados anteriores foram utilizados para o treinamento. O objetivo dessa etapa foi mensurar

o desempenho preditivo do modelo, respeitando a ordem cronológica dos dados e prevenindo vazamento de informações entre as etapas.

As métricas utilizadas para a avaliação foram o Erro Quadrático Médio (MSE), a Raiz do Erro Quadrático Médio (RMSE), o Coeficiente de Determinação (R^2) e o Erro Percentual Absoluto Médio (MAPE). O MAPE foi definido como métrica principal de otimização, por oferecer uma medida relativa do erro em relação à magnitude dos valores previstos, sendo amplamente empregada em estudos de previsão de séries temporais (HYNDMAN; KOEHLER, 2006). As demais métricas foram utilizadas de forma complementar, permitindo uma avaliação mais abrangente da precisão e estabilidade do modelo.

Durante a otimização dos hiperparâmetros, foi utilizada validação cruzada temporal com dois *splits* (*TimeSeriesSplit*), a fim de garantir uma avaliação consistente e evitar tanto o sobreajuste quanto o subajuste. Após a escolha do melhor conjunto de parâmetros, o modelo final foi treinado integralmente sobre os dados de treino e posteriormente avaliado no conjunto de teste. Os resultados quantitativos obtidos nas quatro métricas estão apresentados na seção de resultados.

4.6 DELIMITAÇÕES

Os dados utilizados neste projeto abrangem o período de 2022 ao primeiro quadrimestre de 2025, com frequência quadrimestral. Essa escolha deve-se à forma como os indicadores são disponibilizados no portal do SISAB, onde as informações estão organizadas por quadrimestres nesse período de tempo (SISAB, 2025). Além disso, este estudo se restringe à utilização de indicadores que estejam disponíveis publicamente e possuam cobertura nacional.

Outra delimitação refere-se ao treinamento do modelo exclusivamente com municípios de grande porte, uma vez que municípios menores apresentam variações mais acentuadas e valores inconsistentes, o que pode comprometer o desempenho e a avaliação do modelo preditivo.

5. RESULTADO E DISCUSSÃO

Neste capítulo são apresentados os resultados obtidos a partir das análises realizadas, abrangendo desde a construção do processo de ETL e exploração inicial dos dados até a avaliação do modelo preditivo. O objetivo é examinar o comportamento das séries temporais de mortalidade neonatal e discutir o desempenho do modelo desenvolvido, considerando sua precisão, estabilidade e adequação ao contexto da saúde pública brasileira.

A primeira seção descreve o pipeline de ETL, detalhando sua estrutura, desempenho e etapas de integração entre as bases SINASC, SIM e SISAB, que viabilizaram a consolidação dos dados em um formato adequado à modelagem preditiva.

A segunda parte contempla uma análise exploratória dos dados, com o intuito de identificar padrões temporais, tendências e sazonalidades relevantes para a modelagem. Em seguida, são apresentados os resultados referentes à modelagem preditiva, incluindo o processo de otimização do modelo XGBoost utilizando a otimização Bayesiana e a comparação com o modelo estatístico tradicional SARIMA, utilizado como baseline para avaliar os ganhos obtidos com a abordagem multivariada de aprendizado de máquina.

Por fim, são discutidos os principais achados do estudo, destacando o potencial da modelagem proposta como uma ferramenta para a predição da taxa de mortalidade neonatal, possibilitando políticas públicas mais ágeis e eficazes.

5.1 PROCESSO DE ETL

O processo de ETL desenvolvido neste trabalho apresentou desempenho satisfatório na extração e integração das bases de dados do SINASC, SIM e SISAB, adaptando-se adequadamente às diferentes formas de disponibilização de cada plataforma. O destaque fica para o SISAB, cuja estrutura de acesso exigiu o desenvolvimento de um *web scraper* dedicado para realizar o download automatizado das informações.

O código completo do ETL, disponível na plataforma Códigos UFSC (https://codigos.ufsc.br/tcc_etl_modelo_multivariado_previsao_mortalidade/etl_tcc/-/tree/763a67c1321ddaf6c8fbd27d044356c3c6604185/) e na seção de apêndices,

contém os parâmetros que permitem definir o intervalo temporal de extração e o processamento dos dados de forma dinâmica. Essa flexibilidade possibilita a atualização periódica das informações sem necessidade de alterações estruturais no script.

Os dados tratados são armazenados no Google BigQuery, garantindo escalabilidade e facilidade de acesso durante as etapas de análise e modelagem. No entanto, o sistema foi desenvolvido de forma modular, permitindo a substituição do banco de dados de destino com alterações mínimas, restritas apenas à etapa de carregamento, uma vez que as fases de extração e transformação são independentes do repositório utilizado.

Outro ponto de destaque é o formato de armazenamento adotado. O ETL desenvolvido carrega os registros em tabelas separadas, mantendo um registro por linha, sem agrupamentos prévios por município, data ou outro nível de agregação. Esse formato foi escolhido para garantir maior flexibilidade analítica, permitindo que diferentes usuários possam consultar, filtrar e agrupar os dados conforme as necessidades de cada estudo ou aplicação futura.

5.2 ANÁLISE EXPLORATÓRIA DOS DADOS

Nesta etapa, o objetivo foi compreender o padrão dos dados, sua disposição ao longo do tempo e a qualidade geral das informações utilizadas no estudo. Inicialmente, foram realizadas análises descritivas e contagem de valores nulos, a fim de avaliar a consistência dos registros provenientes das bases do SINASC, SIM e SISAB.

A Tabela 2 apresenta as estatísticas descritivas das principais variáveis utilizadas, incluindo os indicadores assistenciais do SISAB e a taxa de mortalidade neonatal.

Tabela 2 – Estatísticas descritivas dos indicadores da Atenção Primária e da taxa de mortalidade neonatal em cidades de grande porte

Indicador	Média	Mediana	Desvio padrão	Mínimo	Máximo

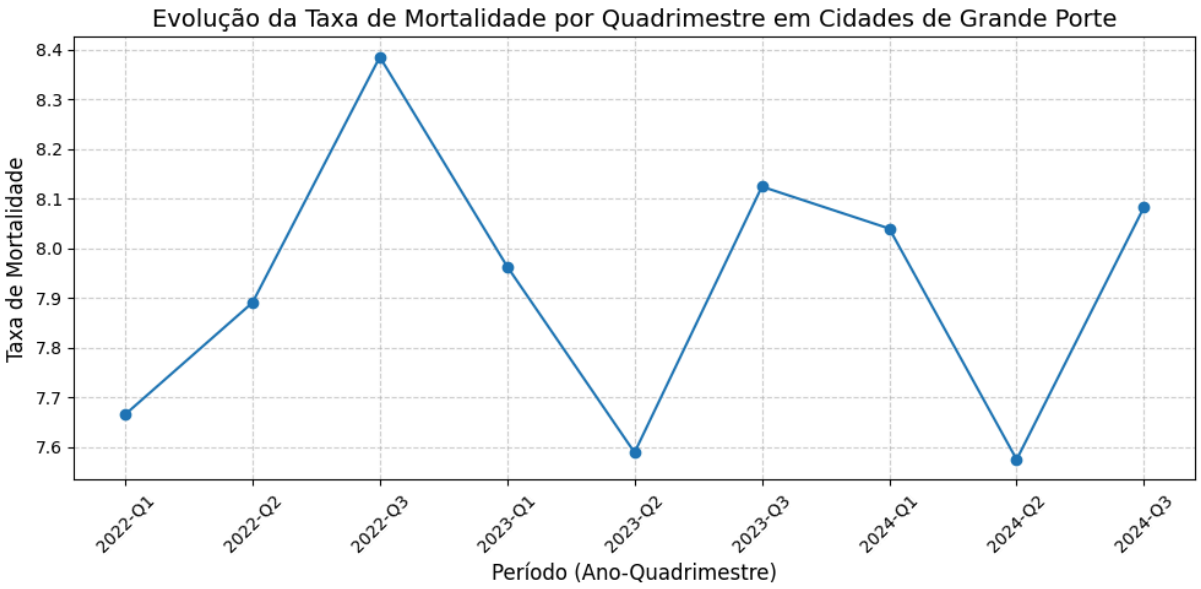
Proporção de gestantes com pelo menos 6 consultas de pré-natal realizadas, sendo a primeira até a 12ª semana de gestação (%)	40,71	41,00	15,33	0,00	87,00
Proporção de gestantes com atendimento odontológico realizado (%)	45,43	46,00	19,15	0,00	96,00
Proporção de gestantes com realização de exames para sífilis e HIV (%)	59,87	60,00	19,22	1,00	99,00
Proporção de mulheres com coleta de citopatológico na Atenção Primária à Saúde (%)	20,78	20,00	8,70	1,00	50,00
Taxa de mortalidade neonatal (óbitos até 28 dias por 1.000 nascidos vivos)	7,92	7,65	2,42	3,75	13,94

Fonte: elaborado pelo autor.

Com base nesses resultados, observa-se que a taxa de mortalidade neonatal apresenta alta heterogeneidade com mínimos e máximos distantes. Já os indicadores do SISAB mostram médias e medianas próximas, porém também com variabilidade considerável evidenciado pelo alto desvio padrão.

A Figura 1 apresenta a evolução temporal da taxa média de mortalidade neonatal no período analisado.

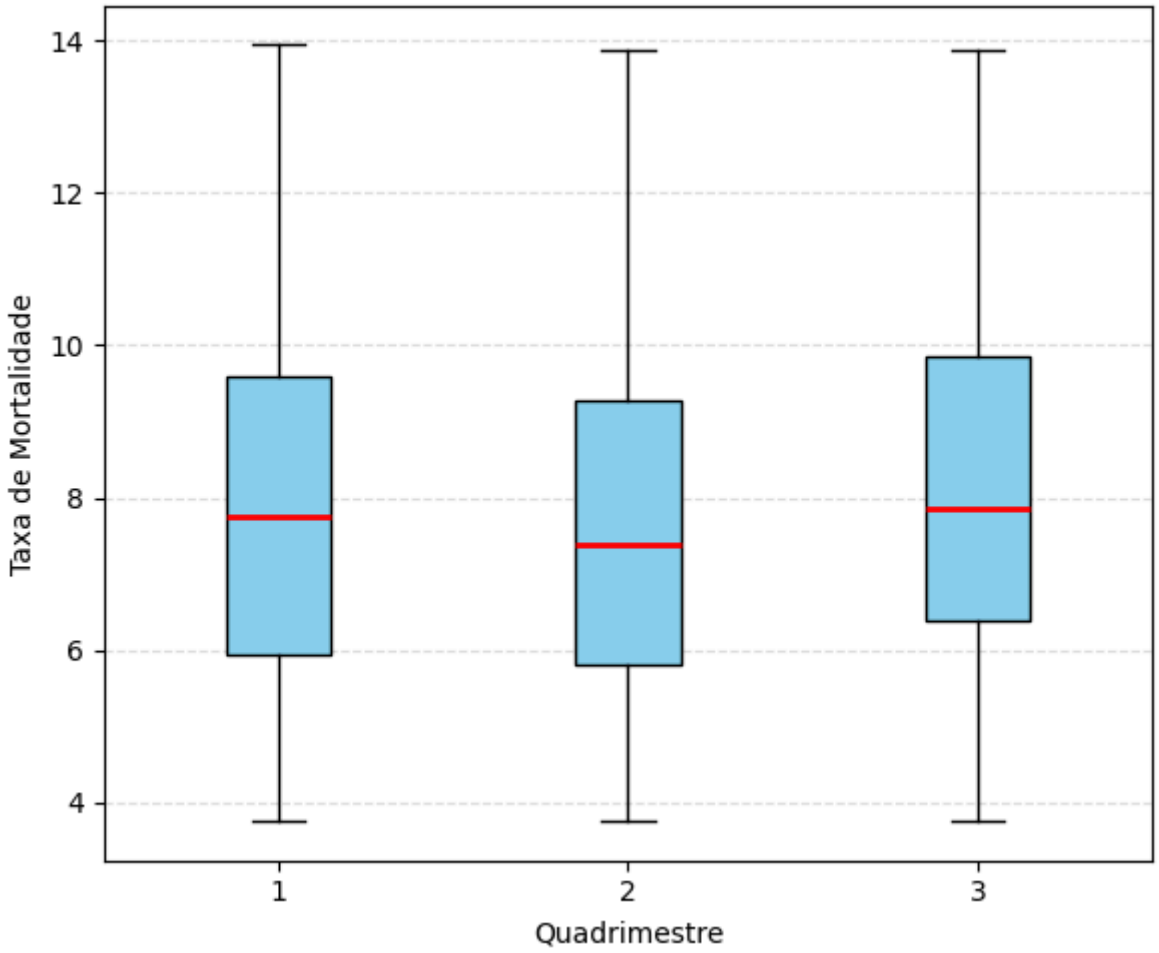
Figura 1 – Evolução quadrimestral da taxa média de mortalidade neonatal entre 2022 e 2024 em Cidades de Grande porte



Fonte: elaborado pelo autor.

É possível observar um padrão sazonal bem definido, com menores valores concentrados no segundo quadrimestre (Q2) e aumento gradual até atingir o pico no terceiro quadrimestre (Q3). Esse comportamento se confirma na Figura 2, que apresenta o boxplot da taxa de mortalidade agrupada por quadrimestre, onde se percebe uma elevação progressiva dos valores medianos do Q2 a Q3.

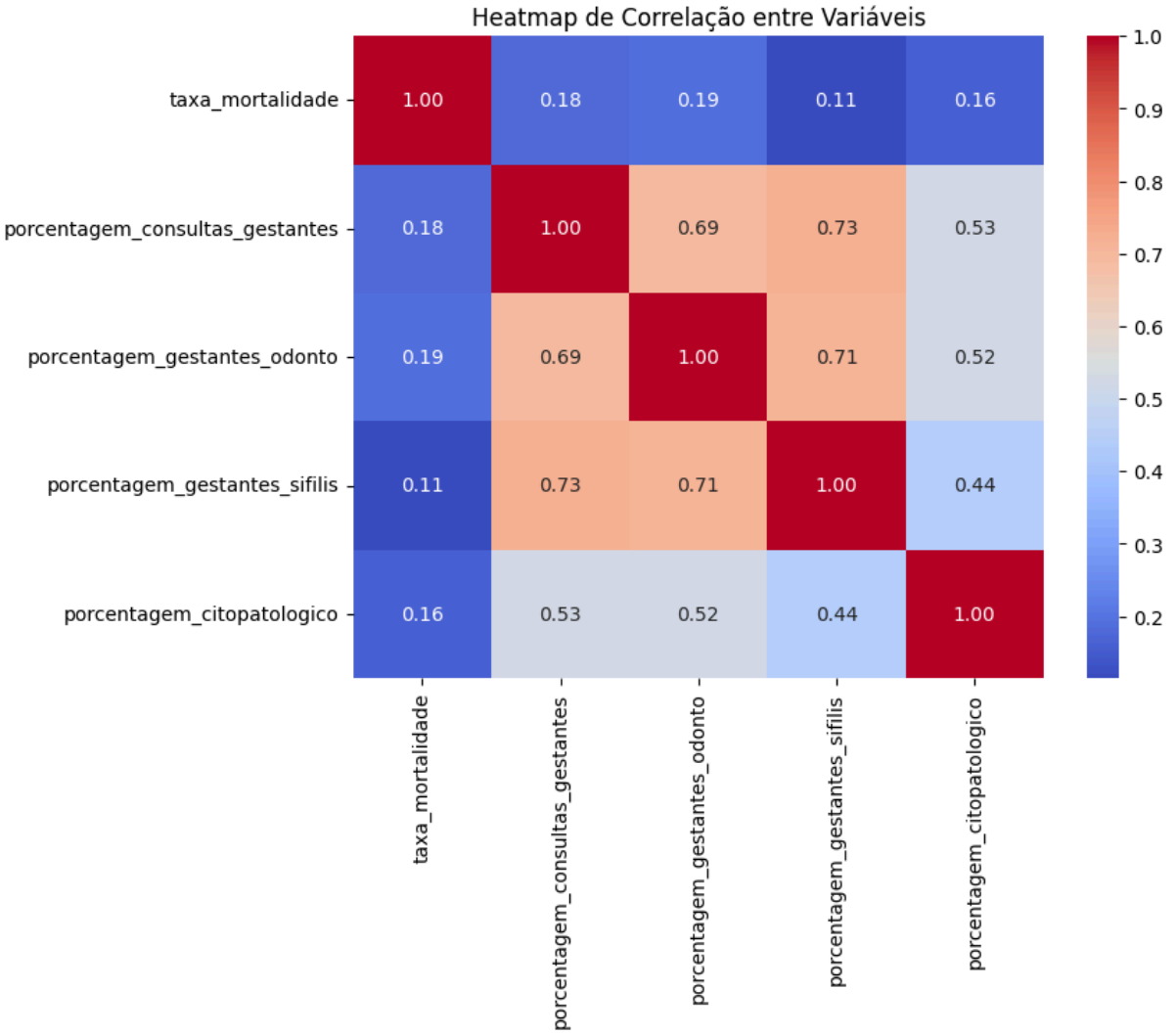
Figura 2 – Boxplot da taxa de mortalidade neonatal por quadrimestre



Fonte: elaborado pelo autor.

Em relação às correlações entre variáveis, verificou-se que os indicadores assistenciais do SISAB possuem forte correlação entre si, com coeficientes geralmente superiores a 0,6. No entanto, a correlação entre esses indicadores e a taxa de mortalidade neonatal é baixa, variando de 0,13 a 0,18, o que indica que a relação entre cobertura assistencial e mortalidade não é linear simples, exigindo modelos multivariados mais sofisticados para capturar tais interações.

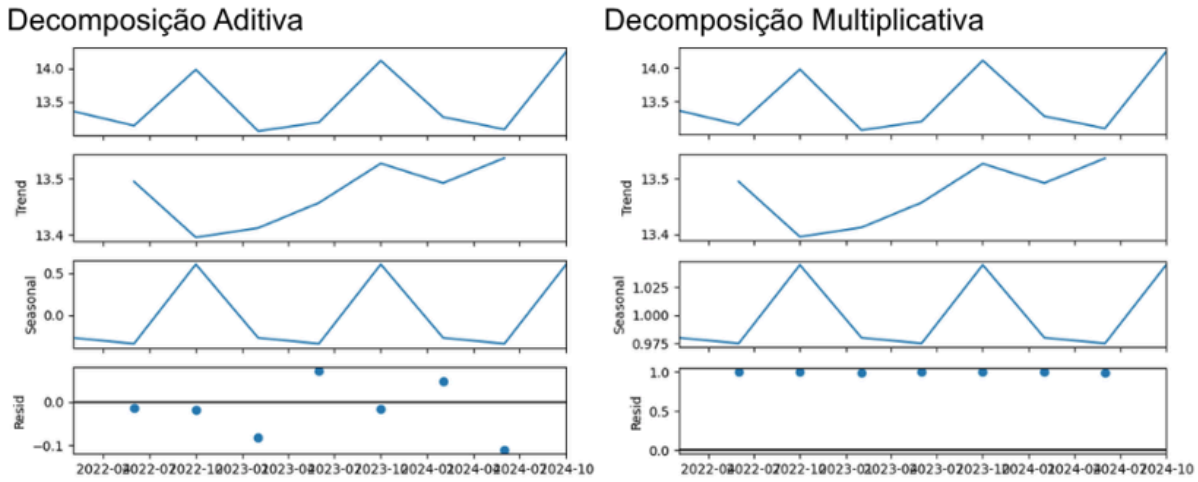
Figura 3 – Matriz de correlação entre indicadores do SISAB e taxa de mortalidade neonatal



Fonte: elaborado pelo autor.

Por fim, foi realizada a decomposição da série temporal pelos métodos aditivo e multiplicativo, com o objetivo de identificar os componentes de tendência, sazonalidade e ruído. Os resultados indicaram que a série é não estacionária, apresentando um padrão sazonal bem definido e um resíduo não aleatório indicando informações estruturadas nos dados não explicadas apenas pela tendência e sazonalidade.

Figura 4 – Decomposição da série temporal: componentes de tendência, sazonalidade e resíduo



Fonte: elaborado pelo autor.

Essas análises exploratórias evidenciam a presença de dependência temporal e sazonalidade consistente na série, o que justifica a adoção de técnicas de modelagem capazes de capturar relações dinâmicas entre os indicadores e a taxa de mortalidade.

Os padrões identificados reforçam a hipótese de que as variações da mortalidade neonatal refletem tanto a sazonalidade demográfica quanto às diferenças na cobertura assistencial. Com base nessas evidências, procedeu-se à etapa de modelagem, buscando capturar essas relações temporais e avaliar a capacidade preditiva do modelo proposto.

5.3 MODELAGEM DE DADOS E OTIMIZAÇÃO

Antes da etapa de otimização, foi conduzida uma análise comparativa preliminar com o objetivo de identificar qual modelo apresentava melhor adequação ao padrão dos dados. Para isso, foram avaliados três modelos distintos: XGBoost, SARIMAX e Regressão Linear Múltipla, todos inicialmente aplicados sem ajuste de hiperparâmetros

A Tabela 3 apresenta a comparação dos resultados obtidos entre os modelos. Com base nas métricas, o XGBoost apresentou o melhor equilíbrio entre precisão, robustez e estabilidade, sendo selecionado como o modelo final deste estudo.

Tabela 3 – Métricas de erro dos modelos testados: Regressão Linear Múltipla, SARIMA, XGBoost versão não otimizada

Modelo	MSE	RMSE	R^2	MAPE
XGBoost	5.84	2.4	-0.03	0.28
Sarimax	5.79	2.4	-0.02	0.30
Regressão Linear Múltipla	5.92	2.4	-0.04	0.29

Fonte: elaborado pelo autor.

Ao final do processo, o modelo otimizado apresentou desempenho superior nas métricas MAPE, RMSE e R^2 em relação aos demais algoritmos testados, evidenciando sua capacidade de generalização e adequação à natureza dos dados. A configuração final do modelo foi então utilizada para realizar as previsões sobre os períodos de teste, cujos resultados são apresentados na seção de avaliação.

Com os dados consolidados, foi realizada a etapa de otimização e avaliação do modelo XGBoost, utilizando busca Bayesiana para definição dos hiperparâmetros. O espaço de busca e os melhores valores encontrados estão apresentados na Tabela 4, que sintetiza os parâmetros ajustados e aqueles que mais contribuíram para o desempenho final do modelo.

Tabela 4 – Hiperparâmetros otimizados via busca bayesiana e respectivos valores selecionados

Hiperparâmetro	**Descrição**	**Valor selecionado**
`colsample_bylevel`	Fração de colunas amostradas em cada nível da árvore	0,30
`colsample_bytree`	Fração de colunas amostradas por árvore	1,00

gamma	Parâmetro de regularização que controla a complexidade do modelo	2,35
learning rate	Taxa de aprendizado (shrinkage) usada para atualização dos pesos	0,001
max_bin	Número máximo de divisões de histogramas usadas no particionamento	512
max_depth	Profundidade máxima das árvores de decisão	30
min_child_weight	Peso mínimo exigido em um nó filho antes da divisão	29
n_estimators	Número total de árvores (iterações de boosting)	4798
reg_alpha	Termo de regularização L1 (Lasso)	8,51
reg_lambda	Termo de regularização L2 (Ridge)	100,00
subsample	Proporção de amostras utilizadas para treinar cada árvore	1,00

Fonte: elaborado pelo autor.

O modelo foi treinado com o conjunto completo de municípios de grande porte, sem segmentação por unidade federativa ou região, garantindo maior robustez estatística no processo de ajuste. Os resultados obtidos para as métricas de avaliação estão dispostos na Tabela 5, com destaque para o MAPE, métrica principal de otimização.

Tabela 5 – Desempenho global do modelo XGBoost: MSE, RMSE, R^2 e MAPE

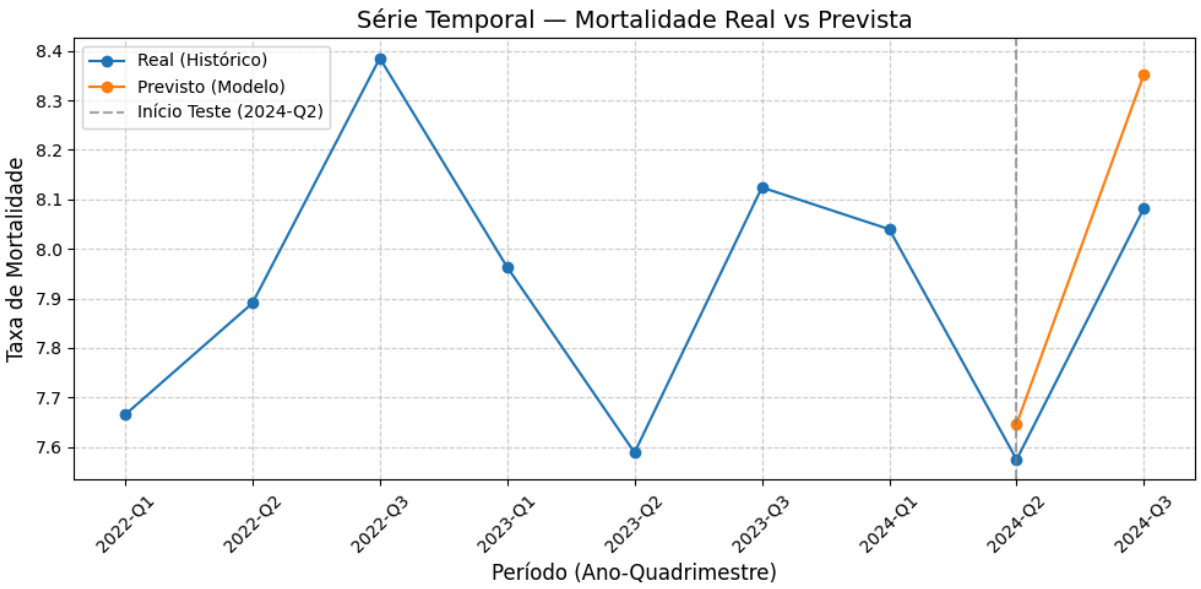
Métrica	Valor obtido
MSE (Mean Squared Error)	5,51
RMSE (Root Mean Squared Error)	2,35
R^2 (Coeficiente de determinação)	0,03
MAPE (Mean Absolute Percentage Error)	0,28

Fonte: elaborado pelo autor.

De forma geral, o modelo apresentou desempenho satisfatório, capturando adequadamente a tendência e a sazonalidade média da série temporal. No entanto, verificou-se que a maior parte das previsões concentra-se na faixa entre 7 e 9 óbitos por mil nascidos vivos, faixa também predominante na amostra. O modelo demonstrou menor capacidade de adaptação para municípios com taxas muito altas ou muito baixas, o que pode ser atribuído à baixa representatividade desses casos extremos no conjunto de treinamento.

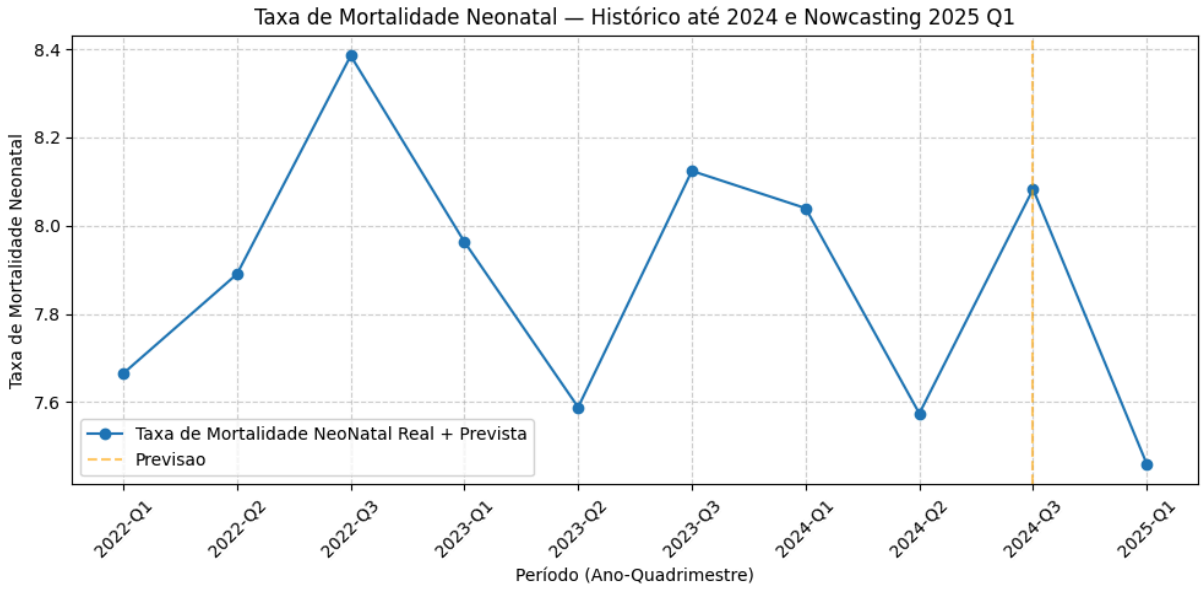
As Figuras 5 e 6 ilustram o comportamento do modelo ao comparar os valores observados e previstos da taxa média de mortalidade neonatal, incluindo a projeção (*nowcasting*) para o primeiro quadrimestre de 2025.

Figura 5 – Série temporal quadrimestral da taxa de mortalidade neonatal: valores preditos para 2024.



Fonte: elaborado pelo autor.

Figura 6 – Série temporal quadrimestral da taxa de mortalidade neonatal: valores previstos para 2025.

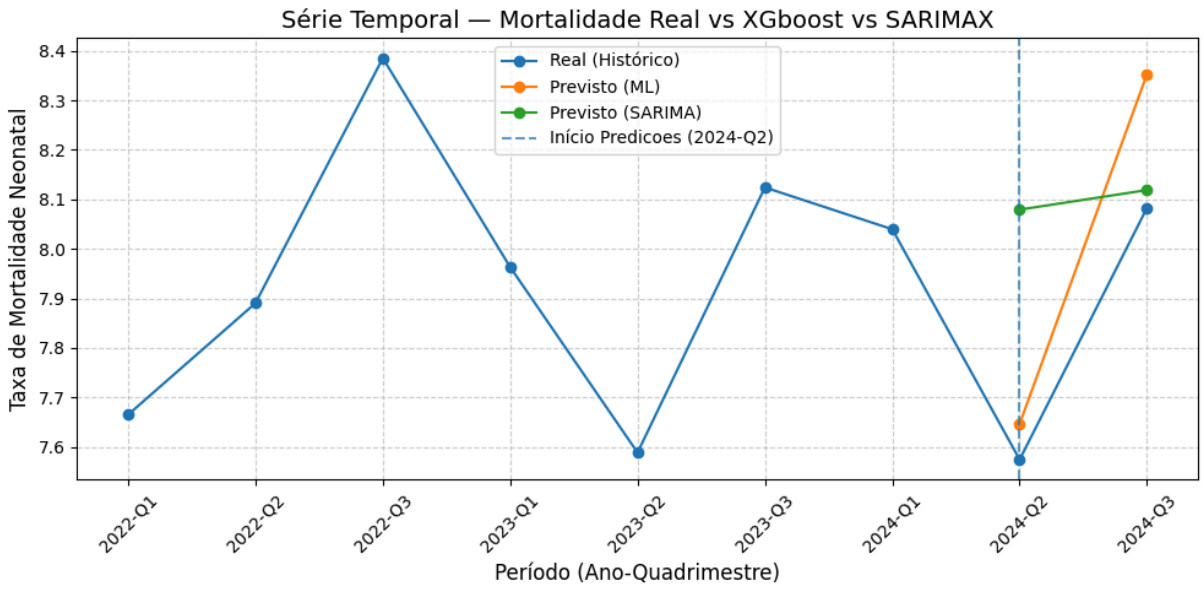


Fonte: elaborado pelo autor.

Nota-se que o modelo reproduz bem as variações quadrimestrais, apresentando boa aderência nas tendências gerais, ainda que tenda a subestimar valores em períodos de maior mortalidade. Esse comportamento reforça o padrão identificado na análise exploratória, em que valores extremos são menos frequentes e, portanto, mais difíceis de prever com precisão.

Na comparação com o modelo SARIMA, observa-se que o modelo estatístico apresenta maior dificuldade em capturar o padrão temporal da série, especialmente em períodos de variação acentuada.

Figura 7 – Série temporal quadrimestral da taxa de mortalidade neonatal comparação Sarima e XGboost



Fonte: elaborado pelo autor.

Esse resultado reforça a adequação da abordagem multivariada baseada em aprendizado de máquina, que demonstrou melhor capacidade de representar as flutuações e a sazonalidade da taxa de mortalidade neonatal.

A Tabela 6 resume as métricas de erro correspondentes, evidenciando o desempenho superior do XGBoost em todas as métricas avaliadas, com destaque para o MAPE.

Tabela 6 – Comparativo de desempenho entre SARIMA e XGBoost: MSE, RMSE, R^2 e MAPE

Métrica	XGboost	Sarima
MSE (Mean Squared Error)	5,51	5.79
RMSE (Root Mean Squared Error)	2,35	2.4

R^2 (Coeficiente de determinação)	0,03	-0.02
MAPE (Mean Absolute Percentage Error)	0,28	0.30

Fonte: elaborado pelo autor.

De forma geral, a etapa de otimização e avaliação confirmou a consistência e estabilidade do modelo XGBoost na previsão da taxa de mortalidade neonatal, justificando sua escolha como modelo principal para as análises comparativas e regionais apresentadas nas próximas seções.

5.4 ANÁLISE DOS RESULTADOS E AVALIAÇÃO DA GENERALIZAÇÃO DO MODELO

Embora o modelo tenha apresentado desempenho global satisfatório, foi conduzida uma análise adicional visando avaliar seu comportamento quando aplicado a dados segregados por Unidade Federativa (UF). O objetivo é verificar em quais regiões o modelo apresenta melhor capacidade de generalização e se havia diferenças significativas de desempenho entre os contextos estaduais.

A análise por UF, entretanto, mostrou-se inconclusiva, em razão da presença de estados com número reduzido de municípios de grande porte, o que comprometeu a representatividade estatística dos resultados. Por esse motivo, optou-se por agrupar as unidades federativas por região, permitindo comparações mais consistentes e interpretáveis.

Nessa análise regional, observaram-se diferenças relevantes no desempenho do modelo. As regiões Nordeste e Centro-Oeste apresentaram valores de erro mais elevados, indicando maior variabilidade entre os municípios. Em contrapartida, as regiões Norte e Sul e, em especial, os estados Roraima, Amapá, Alagoas, Rondônia, Piauí, Paraná, Sergipe, apresentaram taxas de mortalidade neonatal médias entre 7 e 9 óbitos para cada mil nascidos vivos, faixa em que o modelo demonstrou maior estabilidade e precisão preditiva, atingindo erro médio (MAPE) de aproximadamente 0.22 ou 22% de Erro.

A Tabela 7 resume as métricas de erro por região e pelos conjuntos de estados com melhor desempenho, evidenciando a capacidade do modelo em reproduzir as tendências e sazonalidades observadas em cada contexto territorial.

Tabela 7 – Métricas de erro por região e conjuntos de estados

Região	MSE	RMSE	R^2	MAPE
Sul	3.63	1.90	0.006	0.26
Sudeste	5.38	2.32	0.020	0.27
Centro-Oeste	6.74	2.59	-0.099	0.28
Nordeste	7.22	2.68	-0.048	0.28
Norte	4.44	2.10	0.0617	0.26
Conjunto de Melhores Estados (Conjunto com menor erro)	3.53	1.88	0.054	0.22

Fonte: elaborado pelo autor.

Esse comportamento é justificado pelas próprias características estruturais do Brasil. O país possui dimensões continentais e contextos regionais amplamente distintos, com diferenças marcantes em indicadores demográficos, cobertura assistencial e infraestrutura de saúde. Em razão disso, modelos generalistas tendem a perder desempenho em contextos heterogêneos.

Ainda assim, o modelo proposto demonstrou boa capacidade de adaptação em contextos regionais específicos, especialmente em áreas com padrões de mortalidade mais estáveis e indicadores assistenciais consolidados. Esse resultado reforça o potencial da abordagem adotada como uma ferramenta de apoio à gestão regionalizada, permitindo identificar variações locais e antecipar cenários de risco de forma mais precisa e tempestiva.

Os resultados obtidos demonstram que o modelo proposto se mostrou capaz de prever de forma consistente as taxas de mortalidade neonatal, especialmente em

municípios cuja taxa se encontra próxima à média dos municípios de grande porte. Essa faixa concentra a maior parte dos registros com aproximadamente 60% dos valores entre 6,5 a 9,5 óbitos a cada mil nascidos vivos e representa o padrão mais recorrente dos dados analisados, o que explica o bom desempenho observado.

Por outro lado, o modelo apresentou tendência a aproximar valores extremos da média, subestimando taxas muito altas e superestimando taxas muito baixas. Esse comportamento, característico de modelos baseados em gradiente reforçado, decorre da distribuição desigual dos dados de treinamento, em que há escassez de exemplos nos extremos da série. Observou-se que esse efeito é mais pronunciado em regiões como o Sul e o Sudeste, onde predominam municípios com taxas de mortalidade neonatal historicamente mais baixas com os valores mais baixos entre 3,5 a 5,5 com quase 20% dos valores nessa faixa.

Apesar dessas limitações, o desempenho geral pode ser considerado satisfatório, uma vez que o modelo conseguiu capturar as tendências sazonais e temporais da mortalidade neonatal com boa estabilidade. A estrutura quadrimestral adotada favoreceu a detecção de padrões periódicos de variação, tornando o modelo particularmente útil para o acompanhamento contínuo de indicadores em janelas temporais regulares.

Do ponto de vista prático, o modelo desenvolvido tem potencial para apoiar o monitoramento contínuo da mortalidade neonatal, permitindo que gestores públicos acompanhem tendências de forma mais ágil, sem depender da consolidação tardia dos dados oficiais. Essa característica de *nowcasting* pode auxiliar na formulação de respostas rápidas e na priorização de políticas de atenção materno-infantil.

As limitações do modelo estão associadas principalmente ao curto período disponível na base do SISAB (2022–2025), que restringe a variabilidade temporal observada e reduz o potencial de generalização do modelo para intervalos mais longos. Com a expansão da base de dados nos próximos anos, espera-se que o modelo possa ser reajustado e avaliado novamente, de modo a verificar possíveis ganhos de desempenho decorrentes da inclusão de novos períodos de observação.

6 CONCLUSÃO

O presente trabalho teve como objetivo desenvolver um pipeline de ETL e um modelo preditivo multivariado para estimar a taxa de mortalidade neonatal nos municípios brasileiros, utilizando dados provenientes dos sistemas públicos de informação em saúde.O ETL desenvolvido mostrou-se robusto e funcional, conseguindo integrar de forma automatizada as bases do SIM, SINASC e SISAB, mesmo diante das diferenças estruturais entre esses sistemas. A solução implementada garante flexibilidade, permitindo que novos períodos sejam incorporados de maneira simples e reprodutível, além de possibilitar o reaproveitamento do pipeline para futuras aplicações em outras bases de dados de saúde pública.

Em relação à modelagem, o estudo apresentou um desempenho satisfatório, capturando tendências e padrões sazonais de mortalidade neonatal, especialmente nas faixas mais comuns observadas entre os municípios de grande porte. Ainda que o modelo tende a aproximar valores extremos da média, ele demonstrou estabilidade e potencial para apoiar o monitoramento contínuo da taxa de mortalidade neonatal.

De modo geral, os resultados obtidos reforçam a viabilidade da abordagem adotada e evidenciam a importância de ferramentas baseadas em aprendizado de máquina para subsidiar decisões mais ágeis e fundamentadas na gestão pública de saúde.

6.1 SUGESTÕES PARA TRABALHOS FUTUROS

Como perspectivas de continuidade, destacam-se alguns eixos de aprimoramento.

Primeiramente, recomenda-se o aperfeiçoamento do tratamento de dados faltantes, explorando técnicas de imputação estatística ou temporal que minimizem o impacto de lacunas nos registros municipais.

Além disso, propõe-se a inclusão de novas variáveis relacionadas à saúde materna, capazes de enriquecer o conjunto de preditores e melhorar a capacidade explicativa do modelo.

No campo técnico, o pipeline desenvolvido poderá ser expandido para integrar outras bases nacionais, como informações do IBGE ou outros indicadores relacionados à saúde materna, ampliando a contextualização das análises e o potencial de generalização dos resultados.

Também pode-se realizar a criação de dashboards automatizados para visualização interativa das taxas e previsões, facilitando o acesso e a interpretação dos dados por gestores públicos.

Por fim, destaca-se o interesse em estabelecer parcerias com órgãos públicos e instituições de pesquisa, a fim de aplicar o modelo em cenários reais de vigilância e planejamento em saúde.

REFERÊNCIAS

BRASIL. Ministério da Saúde. Previne Brasil: novo modelo de financiamento da Atenção Primária à Saúde. Disponível em: https://www.gov.br/saude/pt-br/composicao/saps/previne-brasil. Acesso em: 29 jun. 2025.

CESAR, Marco; SILVA, D. A. Modelo preditivo dos anos de vida perdidos por morte prematura para os municípios brasileiros de médio porte utilizando aprendizagem de máquina. Trabalho de Conclusão de Curso. Universidade Federal de Santa Catarina, [s.d.]. Disponível em: https://codigos.ufsc.br/marco.cesar/yll_time_series_machine_learning. Acesso em: 29 jun. 2025.

AGUIAR, Wellington Sousa. Desenvolvimento de modelos preditivos de mortalidade infantil com base em inteligência artificial no estado do Ceará. 2019. 163 f. Tese (Doutorado em Saúde Pública) - Faculdade de Medicina. Programa de Pós-Graduação em Saúde Pública, Universidade Federal do Ceará, Fortaleza, 2019. Disponível em: http://www.repositorio.ufc.br/handle/riufc/72867. Acesso em: 29 jun. 2025.

GARCIA, L. P.; SCHNEIDER, I. J. C.; DE OLIVEIRA, C.; et al. What is the impact of national public expenditure and its allocation on neonatal and child mortality? A machine learning analysis. BMC Public Health, v. 23, p. 793, 2023. Disponível em: https://doi.org/10.1186/s12889-023-15683-y. Acesso em: 29 jun. 2025.

MANGOLD, C.; ZORETIC, S.; THALLAPUREDDY, K.; MOREIRA, A.; CHORATH, K.; MOREIRA, A. Machine learning models for predicting neonatal mortality: A systematic review. Neonatology, v. 118, n. 4, p. 394–405, 2021. Disponível em: https://doi.org/10.1159/000516891. Acesso em: 29 jun. 2025.

SISAB – Sistema de Informação em Saúde para a Atenção Básica. Painel de indicadores federais. Disponível em:

https://sisab.saude.gov.br/paginas/acessoRestrito/relatorio/federal/indicadores/indicadorPainel.xhtml. Acesso em: 29 jun. 2025.

HEATON, Jeff. Artificial Intelligence for Humans: Neural Networks and Deep Learning. 3. ed. [S. l.]: Heaton Research, 2015. Disponível em: http://www.mattversaggi.com/mit_open_courseware/Artificial_Intelligence_for_Humans/aifh_v3_neuralnets_ deel_learning.pdf. Acesso em: 29 jun. 2025.

Batista, A. F. M., Diniz, C. S. G., Bonilha, E. A., Kawachi, I., & Chiavegatto Filho, A. D. P. (2021). Neonatal mortality prediction with routinely collected data: a machine learning approach. BMC Pediatrics, 21(1). https://doi.org/10.1186/s12887-021-02788-9

Turner, R., Eriksson, D., Mccourt, M., Kiili, J., Xu, V. Z., Escalante, H. J., & Hofmann, K. (n.d.). Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. http://clopinet.com/isabelle/Projects/NIPS2006/

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. Journal of Machine Learning Research, v. 13, n. 1, p. 281-305, 2012.

COLLINS, G. S.; REITSMA, J. B.; ALTMAN, D. G.; MOONS, K. G. M. Transparent reporting of a multivariable prediction model for individual prognosis or diagnosis (TRIPOD): the TRIPOD statement. BMJ, v. 350, 2015.

GOOGLE CLOUD. BigQuery: Cloud Data Warehouse. 2021. Disponível em: https://cloud.google.com/bigquery. Acesso em: 5 jul. 2025.

HYNDMAN, R. J.; ATHANASOPOULOS, G. Forecasting: principles and practice. 2. ed. Melbourne: OTexts, 2018.

HYNDMAN, R. J.; KOEHLER, A. B. Another look at measures of forecast accuracy. International Journal of Forecasting, v. 22, n. 4, p. 679-688, 2006.

KIMBALL, R.; CASERTA, J. The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data. Indianapolis: Wiley Publishing, 2004.

LAWN, J. E.; BLENCOWE, H.; OZA, S.; YOU, D.; LEE, A. C.; WAISWA, P.; LANZOU, M.; BHUTTA, Z. A.; BARROS, A. J.; CHRISTIAN, P.; MATHERS, C.; COUSENS, S. N. Every newborn: progress, priorities, and potential beyond survival. The Lancet, v. 384, n. 9938, p. 189-205, 2014.

MANGOLD, C.; ZORETIC, S.; THALLAPUREDDY, K.; MOREIRA, A.; CHORATH, K.; MOREIRA, A. Machine learning models for predicting neonatal mortality: A systematic review. Neonatology, v. 118, n. 4, p. 394–405, 2021. Disponível em: https://doi.org/10.1159/000516891. Acesso em: 5 jul. 2025.

MCKINNEY, W. Data structures for statistical computing in Python. In: PROCEEDINGS OF THE 9TH PYTHON IN SCIENCE CONFERENCE, 2010, Austin. Anais... Austin: SciPy, 2010. p. 51-56.

MINISTÉRIO DA SAÚDE. Manual de instruções para o preenchimento da declaração de nascido vivo. Brasília: Ministério da Saúde, 2011.

MINISTÉRIO DA SAÚDE. Previne Brasil: novo modelo de financiamento da Atenção Primária à Saúde. Brasília: Ministério da Saúde, 2019.

RAJKOMAR, A.; DEAN, J.; KOHANE, I. Machine learning in medicine. New England Journal of Medicine, v. 380, n. 14, p. 1347-1358, 2018.

SHICKEL, B.; TIGHE, P. J.; BIHORAC, A.; RASHIDI, P. Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. IEEE Journal of Biomedical and Health Informatics, v. 22, n. 5, p. 1589-1604, 2017.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. Advances in Neural Information Processing Systems, v. 25, p. 2951-2959, 2012.

TASHMAN, L. J. Out-of-sample tests of forecasting accuracy: an analysis and review. International Journal of Forecasting, v. 16, n. 4, p. 437-450, 2000.

UNITED NATIONS. Transforming our world: the 2030 Agenda for Sustainable Development. New York: United Nations, 2015.

WORLD HEALTH ORGANIZATION. Trends in maternal mortality 2000 to 2017: estimates by WHO, UNICEF, UNFPA, World Bank Group and the United Nations Population Division. Geneva: World Health Organization, 2019.

BHASKARAN, K.; GASPARRINI, A.; HENEGHAN, C.; ARMSTRONG, B. Time series regression studies in environmental epidemiology. *International Journal of Epidemiology*, 2013. Disponível em: https://pdfs.semanticscholar.org/eab4/8c95e22f65938a0d4b590fff5704b5fef0b3.pdf. Acesso em: 5 out. 2025.

BOX, G. E. P.; JENKINS, G. M.; REINSEL, G. C. *Time Series Analysis: Forecasting and Control.* 5. ed. New Jersey: Wiley, 2015.

CHECHI, L.; BAYER, F. M. Modelos univariados de séries temporais para previsão das temperaturas médias mensais de Erechim, RS. *Revista Brasileira de Engenharia Agrícola e Ambiental*, 2012. Disponível em: https://doi.org/10.1590/S1415-43662012001200009. Acesso em: 5 out. 2025.

SHUMWAY, R. H.; STOFFER, D. S. *Time Series Analysis and Its Applications: With R Examples.* 4. ed. New York: Springer, 2017.

MONTGOMERY, D. C.; PECK, E. A.; VINING, G. G. *Introduction to Linear Regression Analysis.* 5. ed. New Jersey: Wiley, 2015.

PALIARI, I.; KARANIKOLA, A.; KOTSIANTIS, S. A comparison of the optimized LSTM, XGBOOST and ARIMA in time series forecasting. In: *2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA)*, Chania, Crete, Greece, 2021. Disponível em: https://ieeexplore.ieee.org/document/9555520. Acesso em: 28 set. 2025

BRASIL. Ministério da Saúde. Secretaria de Vigilância em Saúde. Portaria nº 116, de 11 de fevereiro de 2009. Regulamenta a coleta de dados, fluxo e periodicidade de envio das informações sobre óbitos e nascidos vivos para os Sistemas de Informações em Saúde sob gestão da Secretaria de Vigilância em Saúde. Diário Oficial da União, Brasília, DF, 11 fev. 2009. Disponível em: https://bvsms.saude.gov.br/bvs/saudelegis/svs/2009/prt0116_11_02_2009.html Acesso em: 8 nov. 2025..

APÊNDICE A - CÓDIGO DO ETL DESENVOLVIDO

O Repositório com todos os códigos desenvolvidos e sua respectiva estrutura pode ser acessado pelo link a abaixo: https://codigos.ufsc.br/tcc_etl_modelo_multivariado_previsao_mortalidade.

main.py

```
from transform import *
from load import *
from extract import *
from scrapper import *

def etl_nascidos_vivos(file_key):
    dataset_name = "mortalidade_infantil"
    data_folder_mortalidade = "DadosNascidosVivos"

    # Conexão com GCP
    client = gcp_connection(file_key)
    # Verificar se o dataset já existe, se não existe, cria
    dataset_fonte = dataset_exist(client,dataset_name)
    # Verifica se as tabelas já existem, se não existe, cria
                                    table_nascidos_vivos        =
table_exist_natalidade(client,dataset_fonte)
```

```
    download_files_natalidade(data_folder_mortalidade)

                                    df_nascidos_vivos         =
create_df_natalidade(data_folder_mortalidade)

    # Incluir tabelas e dfs em uma biblioteca
    tables_dfs = {table_nascidos_vivos:df_nascidos_vivos}
    load_data(tables_dfs,client,dataset_fonte)
    return df_nascidos_vivos

def etl_mortalidade(file_key):
    dataset_name = "mortalidade_infantil"
    data_folder_mortalidade = "DadosMortalidade"

    client = gcp_connection(file_key)
    # Verificar se o dataset já existe, se não existe, cria
    dataset_fonte = dataset_exist(client, dataset_name)
    # Verifica se as tabelas já existem, se não existe, cria
         table_mortalidade  =  table_exist_mortalidade(client,
dataset_fonte)

    download_files_mortalidade(data_folder_mortalidade)

                                       df_mortalidade         =
create_df_mortalidade(data_folder_mortalidade)

    tables_dfs = {table_mortalidade: df_mortalidade}
    load_data(tables_dfs, client, dataset_fonte)

def etl_sisab(file_key):
    dataset_name = "dados_sisab"
    data_folder_sisab = "DadosSisab"

    client = gcp_connection(file_key)
    # Verificar se o dataset já existe, se não existe, cria

    download_data_sisab()

    for arquivo in os.listdir(data_folder_sisab):
        dataset_fonte = dataset_exist(client, dataset_name)
          # Verifica se as tabelas já existem, se não existe,
cria
                    tabela_sisab   =   table_exist_sisab(client,
dataset_fonte, arquivo)

        df = processa_df_sisab(arquivo, data_folder_sisab)
        tables_dfs = {tabela_sisab:df}
        load_data(tables_dfs, client, dataset_fonte)


if __name__ == "__main__":
```

```
    file_key = "keys/tcc-pedro-461615-9f18521badd1.json"
    etl_nascidos_vivos(file_key)
    etl_mortalidade(file_key)
    etl_sisab(file_key)
    print("Todos os arquivos baixados:")
```

extract.py

```
# importar bibliotecas
import os
from urllib import request

def download_files_mortalidade(data_folder):
    # Garante que a pasta existe
    os.makedirs(data_folder, exist_ok=True)

    # Baixar os arquivos dos anos desejados
    for ano in range(2021, 2025):
        ano_str = str(ano)


print("-----------------------------------------------")
        print(f"Próximo ano a carregar: {ano_str}")

print("-----------------------------------------------")

        # Definir nome e links
        filename_primary = f"Mortalidade_Geral_{ano_str}.csv"
                    path_primary   =   os.path.join(data_folder,
filename_primary)
                                                    url_primary     =
f"https://diaad.s3.sa-east-1.amazonaws.com/sim/{filename_prima
ry}"

        # Se já existe, pula
        if os.path.exists(path_primary):
                    print(f"Arquivo  {filename_primary}  já  foi
baixado")
            continue

        # Primeiro link
        try:
            print(f"Baixando (primário): {filename_primary}")
            request.urlretrieve(url_primary, path_primary)

        except Exception as e_primary:
            print(f"Falha no primário: {e_primary}")
            # Tenta fallback
            try:
                ano2 = str(ano)[-2:]  # últimos 2 dígitos
                filename_fallback = f"DO{ano2}OPEN.csv"
                     path_fallback = os.path.join(data_folder,
filename_primary)
                                                      url_fallback   =
f"https://s3.sa-east-1.amazonaws.com/ckan.saude.gov.br/SIM/{fi
lename_fallback}"

                                    print(f"Baixando  (fallback):
{filename_fallback}")
```

```
                                request.urlretrieve(url_fallback,
path_fallback)

                if os.path.exists(path_fallback):
                                print(f"Arquivo {filename_fallback}
baixado com sucesso")
                else:
                      print("Não foi possível baixar o arquivo
de fallback.")

            except Exception as e_fallback:
                print(f"Falha no fallback: {e_fallback}")
                       print(f"Arquivos de {ano_str} ainda não
disponibilizados")

print("-------------------------------------------------")
                    print("Todos os arquivos disponíveis foram
baixados!")

print("-------------------------------------------------")
                continue

        # Confirma se baixou (primário ou fallback)
                        if   os.path.exists(path_primary)   and
os.path.getsize(path_primary) > 0:
            print(f"Arquivo {filename_primary} baixado")
                     elif   'path_fallback'   in   locals()   and
os.path.exists(path_fallback):
            print(f"Arquivo {filename_fallback} baixado")
        else:
               print("Não foi possível confirmar o download do
arquivo.")


def download_files_natalidade(data_folder):
    # Garante que a pasta existe
    os.makedirs(data_folder, exist_ok=True)

    # Baixar os arquivos dos anos desejados
    for ano in range(2022, 2025):
        ano_baixar = str(ano)


print("-------------------------------------------------")
        print(f"Próximo ano a carregar: {ano_baixar}")

print("-------------------------------------------------")

        # Nomes e URLs
        filename = f"SINASC_{ano_baixar}.csv"
        dest_path = os.path.join(data_folder, filename)
```

```
                                            url_primary          =
f"https://diaad.s3.sa-east-1.amazonaws.com/sinasc/{filename}"
                                                 url_fallback     =
f"https://s3.sa-east-1.amazonaws.com/ckan.saude.gov.br/SINASC/
csv/{filename}"

        # Se já existe, não baixa novamente
        if os.path.exists(dest_path):
            print(f"Arquivo {filename} já foi baixado")
            continue

        # Tenta baixar da URL primária
        try:
            print(f"Baixando (primário): {filename}")
            request.urlretrieve(url_primary, dest_path)

        except Exception as e_primary:
            print(f"Falha no primário: {e_primary}")
            print("Tentando URL de fallback...")

            # Tenta baixar da URL de fallback
            try:
                request.urlretrieve(url_fallback, dest_path)
            except Exception as e_fallback:
                print(f"Falha no fallback: {e_fallback}")
                    print(f"Arquivos de {ano_baixar} ainda não
disponibilizados")

print("-------------------------------------------------")
                    print("Todos os arquivos disponíveis foram
baixados!")

print("-------------------------------------------------")
                continue  # vai para o próximo ano

        # Verifica se o arquivo foi baixado com sucesso
                          if   os.path.exists(dest_path)   and
os.path.getsize(dest_path) > 0:
            print(f"Arquivo {filename} baixado")
        else:

print("-------------------------------------------------------
")
             print("Não foi possível baixar o arquivo. Execução
finalizada!")

print("-------------------------------------------------------
")
```

load.py

```
from google.cloud import bigquery
from google.oauth2 import service_account
import os
import re


def gcp_connection(file_key):

##################################################################
############
     #                                    Cria a conexão com o GCP
#

##################################################################
############


print("##################################################################
####################")
     print("#                                  Iniciando execução do
programa                       #")

print("##################################################################
####################")

print("------------------------------------------------------------
--------------------")
    print("Criando conexão com o GCP...")
    try:
                                           current_directory     =
os.path.dirname(os.path.abspath(__file__))
        file_path = os.path.join(current_directory, file_key)
                                                  credentials     =
service_account.Credentials.from_service_account_file(file_pat
h)
            client  =  bigquery.Client(credentials=credentials,
project=credentials.project_id)
          print(f"Conexão realizada com sucesso com o projeto
{credentials.project_id}.")

print("------------------------------------------------------------
--------------------")
    except Exception:
           print(f"Não foi possível efetivar a conexão com o
GCP.")

print("------------------------------------------------------------
--------------------")
    return client

def dataset_exist(client,dataset_name):
```

```
##################################################################
############
     #                            Cria o dataset caso não exista
#

##################################################################
############

print("-------------------------------------------------------
--------------------")
    print("Verificando a existência do dataset no GCP...")
    dataset_fonte = client.dataset(dataset_name)
    try:
        client.get_dataset(dataset_fonte)
         print(f"O conjunto de dados {dataset_fonte} já existe
no GCP.")

print("-------------------------------------------------------
--------------------")
    except Exception:
         print(f"Dataset {dataset_fonte} não foi encontrado no
GCP, criando o dataset...")
        client.create_dataset(dataset_fonte)
         print(f"O conjunto de dados {dataset_fonte} foi criado
no GCP com sucesso.")

print("-------------------------------------------------------
--------------------")
    return dataset_fonte

def table_exist_natalidade(client,dataset_fonte):

##################################################################
############
     #                          Cria as tabelas caso não existam
#

##################################################################
############

    # Tabela e schema da table_nascidos_vivos
                                    table_nascidos_vivos        =
dataset_fonte.table("nascidos_vivos")

    schema_nascidos_vivos = [
                  bigquery.SchemaField("cd_mun_res",   "STRING",
mode="REQUIRED"),
                        bigquery.SchemaField("ano",   "STRING",
mode="REQUIRED"),
```

```
                        bigquery.SchemaField("quad",   "STRING",
mode="REQUIRED"),
                         bigquery.SchemaField("mes",   "STRING",
mode="REQUIRED"),
               bigquery.SchemaField("total_nascidos",  "FLOAT",
mode="REQUIRED"),
    ]


print("-------------------------------------------------------
--------------------")
    print("Verificando a existência das tabelas no GCP...")
    try:
        client.get_table(table_nascidos_vivos, timeout=30)
        print(f"A tabela {table_nascidos_vivos} já existe!")

print("-------------------------------------------------------
--------------------")
    except:
          print(f"Tabela {table_nascidos_vivos} não encontrada!
Criando tabela {table_nascidos_vivos}...")

client.create_table(bigquery.Table(table_nascidos_vivos,
schema=schema_nascidos_vivos))
        print(f"A tabela {table_nascidos_vivos} foi criada.")

print("-------------------------------------------------------
--------------------")

    return table_nascidos_vivos


def load_data(tables_dfs,client,dataset_fonte):

print("-------------------------------------------------------
--------------------")
    print("Carregando dados no GCP...")
    for tabela, df in tables_dfs.items():
                                                table_ref     =
client.dataset(dataset_fonte.dataset_id).table(tabela.table_id
)
        job_config = bigquery.LoadJobConfig()
                                job_config.write_disposition    =
bigquery.WriteDisposition.WRITE_TRUNCATE
         job = client.load_table_from_dataframe(df, table_ref,
job_config=job_config)
        job.result()
        print(f"Dados carregados na tabela {tabela}.")
```

```
print("-------------------------------------------------------
-------------------")

print("#########################################################
###################")
    print("#                                Dados carregados no GCP
#")

print("#########################################################
###################")


def table_exist_mortalidade(client,dataset_fonte):

#############################################################
############
    #                          Cria as tabelas caso não existam
#

#############################################################
############

    # Tabela e schema da table_mortalidade
                                    table_mortalidade         =
dataset_fonte.table("mortalidade_infantil")

    schema_mortalidade = [
                    bigquery.SchemaField("ano_obito",   "STRING",
mode="REQUIRED"),
                         bigquery.SchemaField("quad",   "STRING",
mode="REQUIRED"),
                       bigquery.SchemaField("dt_obito",   "DATE",
mode="REQUIRED"),
                        bigquery.SchemaField("dt_nasc",   "DATE",
mode="REQUIRED"),
                         bigquery.SchemaField("idade",   "FLOAT",
mode="REQUIRED"),
                    bigquery.SchemaField("cd_mun_res",  "STRING",
mode="REQUIRED"),
                    bigquery.SchemaField("populacao",  "INTEGER",
mode="REQUIRED")
    ]


print("-------------------------------------------------------
-------------------")
    print("Verificando a existência das tabelas no GCP...")
    try:
        client.get_table(table_mortalidade, timeout=30)
        print(f"A tabela {table_mortalidade} já existe!")
```

```
print("-------------------------------------------------------
-------------------")
    except:
            print(f"Tabela {table_mortalidade} não encontrada!
Criando tabela {table_mortalidade}...")
          client.create_table(bigquery.Table(table_mortalidade,
schema=schema_mortalidade))
        print(f"A tabela {table_mortalidade} foi criada.")

print("-------------------------------------------------------
-------------------")

    return table_mortalidade


def table_exist_sisab(client,dataset_fonte, name_table):

############################################################
############
     #                          Cria as tabelas caso não existam
#

############################################################
############
    name_table = re.sub(r'[^A-Za-z0-9_]', '_', name_table)
    # Tabela e schema da table_mortalidade
    table_sisab = dataset_fonte.table(name_table)

    schema = [
        bigquery.SchemaField("UF", "STRING", mode="REQUIRED"),
                        bigquery.SchemaField("IBGE",   "STRING",
mode="REQUIRED"),    #  melhor  STRING  para  manter  zeros  à
esquerda
                    bigquery.SchemaField("Municipio",  "STRING",
mode="REQUIRED"),  # sem acento para evitar problemas
        bigquery.SchemaField("data", "DATE", mode="REQUIRED"),
                  bigquery.SchemaField("porcentagem",  "INTEGER",
mode="REQUIRED")
    ]


print("-------------------------------------------------------
-------------------")
    print("Verificando a existência das tabelas no GCP...")
    try:
        client.get_table(table_sisab, timeout=30)
        print(f"A tabela {table_sisab} já existe!")

print("-------------------------------------------------------
-------------------")
```

```
    except:
          print(f"Tabela {table_sisab} não encontrada! Criando
tabela {table_sisab}...")
               client.create_table(bigquery.Table(table_sisab,
schema=schema))
        print(f"A tabela {table_sisab} foi criada.")

print("-------------------------------------------------------
-------------------")

    return table_sisab
```

scrapper.py

```
import os
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from   selenium.webdriver.support.ui   import   WebDriverWait,
Select
from selenium.webdriver.support import expected_conditions as
EC

def download_data_sisab(download_dir=None):
    """
      Faz o download automático dos indicadores do SISAB em
nível Município.
    Baixa e renomeia os arquivos para facilitar uso posterior.

    Args:
          download_dir (str): pasta de destino para salvar os
downloads.
                                    Se None, usa ./downloads no
diretório atual.

    Returns:
        list: caminhos completos dos arquivos baixados.
    """
    # ===== CONFIGURAÇÕES =====
    if download_dir is None:
        download_dir = os.path.join(os.getcwd(), "DadosSisab")
    os.makedirs(download_dir, exist_ok=True)

    options = webdriver.ChromeOptions()
    prefs = {
        "download.default_directory": download_dir,
        "download.prompt_for_download": False,
        "download.directory_upgrade": True,
        "safebrowsing.enabled": True
    }
    options.add_experimental_option("prefs", prefs)

    driver = webdriver.Chrome(options=options)

driver.get("https://sisab.saude.gov.br/paginas/acessoRestrito/
relatorio/federal/indicadores/indicadorPainel.xhtml")

    wait = WebDriverWait(driver, 30)

    # Indicadores desejados
    indicadores = {
        "10": "gestantes_6_consultas",
        "20": "gestantes_sifilis_hiv",
        "30": "gestantes_odonto",
        "40": "citopatologico"
```

```
    }

    def wait_for_download(before_files):
          """Espera até aparecer um novo arquivo na pasta de
downloads"""
        timeout = 90
        start = time.time()
        while time.time() - start < timeout:
            after_files = set(os.listdir(download_dir))
            new_files = after_files - before_files
            if new_files:
                return new_files.pop()
            time.sleep(1)
          raise TimeoutError("Download não detectado dentro do
tempo limite")

    arquivos_baixados = []

    for value, nome_arquivo in indicadores.items():
        # 1) Selecionar o indicador
                                          select_indicador     =
Select(wait.until(EC.presence_of_element_located((By.ID,
"coIndicador"))))
        select_indicador.select_by_value(value)
        time.sleep(3)

        # 2) Selecionar Município
                                             select_nivel      =
Select(wait.until(EC.presence_of_element_located((By.ID,
"selectLinha"))))
        select_nivel.select_by_value("ibge")
        time.sleep(3)

        # 3) Clicar no botão de download
        before_files = set(os.listdir(download_dir))
                                           botao_download      =
wait.until(EC.element_to_be_clickable((By.XPATH,
"//button[contains(., 'Download')]")))
        botao_download.click()
        time.sleep(3)

        # 4) Selecionar CSV (segunda opção)
                                                opcao_csv      =
wait.until(EC.element_to_be_clickable((By.XPATH,
"//ul[@class='dropdown-menu']/li[2]/a")))
        before_files = set(os.listdir(download_dir))
        opcao_csv.click()

        # 5) Esperar download terminar
        arquivo = wait_for_download(before_files)
        time.sleep(10)
```

```
        # 6) Renomear arquivo
            arquivos_atuais = set(os.listdir(download_dir)) -
before_files
        if not arquivos_atuais:
                raise FileNotFoundError("Download não finalizou
corretamente.")
        arquivo_final = arquivos_atuais.pop()

        # Renomear
        old_path = os.path.join(download_dir, arquivo_final)
                        new_path = os.path.join(download_dir,
f"{nome_arquivo}.csv")
        os.rename(old_path, new_path)

        arquivos_baixados.append(new_path)
        time.sleep(3)
        driver.refresh()
        print(f"Download concluído: {new_path}")

    driver.quit()
    return arquivos_baixados
```

transform.py

```
# importar bibliotecas
import os
import pandas as pd
from calendar import monthrange
from datetime import date

def create_df_natalidade(data_folder):
                                    current_directory        =
os.path.dirname(os.path.abspath(__file__))
    file_path = os.path.join(current_directory, data_folder)


print("-------------------------------------------------------
-------------------")
        print("Carregando  os  dados  dos  arquivos  extraídos,
tratando e concatenando...")

    # lista com os dataframes já tratados
    dfs = []

    # Função para gerar o dataframe
    for arquivo in os.listdir(file_path):
        if arquivo.endswith('.csv'):
            # Ler o arquivo CSV com o pandas
             df = pd.read_csv(os.path.join(file_path, arquivo),
delimiter=';', encoding='ISO-8859-1', low_memory=False)
            df = df[['CODMUNRES','DTNASC']]

            # Realizar transformação das datas de nascimento
                  df['dt_nasc'] = pd.to_datetime(df['DTNASC'],
format='%d%m%Y', errors='coerce')
            # Excluir dados nulos para data de nascimento
            df = df.dropna(subset=['dt_nasc'])

             # Criar as colunas ano_nasc e quadrimestre_nasc e
mes_nasc
                                                    df['ano']    =
df['dt_nasc'].dt.year.astype(float).astype(pd.Int64Dtype()).as
type(str).where(df['dt_nasc'].notna())
                   df['quad'] = pd.cut(df['dt_nasc'].dt.month,
bins=[1, 5, 9, 13], labels=[1, 2, 3], right=False)
                  # Extrair os 6 primeiros dígitos da coluna
CODMUNRES
                                               df['cd_mun_res']    =
df['CODMUNRES'].astype(str).str.slice(stop=6)

            # Selecionar coluna desejadas
            df = df[['ano', 'dt_nasc', 'cd_mun_res', "quad"]]
            # adiciona o dataframe à lista de dataframes
            dfs.append(df)
            print(f'{arquivo} concluido indo para o proximo')
```

```
    # concatena os dataframes em um único dataframe final
    df_group = pd.concat(dfs, ignore_index=True)
       df_natalidade  =  df_group.groupby(['cd_mun_res',  'ano',
'quad']).size().reset_index(name='total_nascidos')


    return df_natalidade


def create_df_mortalidade(data_folder):
                                    current_directory         =
os.path.dirname(os.path.abspath(__file__))
    file_path = os.path.join(current_directory, data_folder)


print("-------------------------------------------------------
-------------------")
        print("Carregando  os  dados  dos  arquivos  extraídos,
tratando e concatenando...")

    # lista com os dataframes já tratados
    dfs = []

    # Função para gerar o dataframe
    for arquivo in os.listdir(file_path):
        if arquivo.endswith('.csv'):
            # Ler o arquivo CSV com o pandas
             df = pd.read_csv(os.path.join(file_path, arquivo),
delimiter=';', encoding='ISO-8859-1', low_memory=False)
             # Realizar transformação das datas de nascimento e
óbito
                df['dt_obito'] = pd.to_datetime(df['DTOBITO'],
format='%d%m%Y', errors='coerce')
                  df['dt_nasc'] = pd.to_datetime(df['DTNASC'],
format='%d%m%Y', errors='coerce')
             # Excluir dados nulos para data de nascimento e de
óbito
            df = df.dropna(subset=['dt_nasc'])
            df = df.dropna(subset=['dt_obito'])
            # Criar a coluna idade em dias
                           df['idade']  =  ((df['dt_obito']  -
df['dt_nasc']).dt.days)
            # Manter apenas dados com idades válidas
            df = df[df['idade'] >= 0]
            # Manter apenas idades menores que 28 dias
            df = df[df['idade'] <= 28]
            # Criar as colunas ano_obito e quadrimestre_obito
                                          df['ano_obito']   =
df['dt_obito'].dt.year.astype(float).astype(pd.Int64Dtype()).a
stype(str).where(df['dt_obito'].notna())
```

```
            df['quad_obito'] = pd.cut(df['dt_obito'].dt.month,
bins=[1, 5, 9, 13], labels=[1, 2, 3], right=False)
                # Extrair os 6 primeiros dígitos da coluna
CODMUNRES
                                            df['cd_mun_res']    =
df['CODMUNRES'].astype(str).str.slice(stop=6)
            # Selecionar coluna desejadas
                                                          df     =
df[['ano_obito','quad_obito','dt_obito','dt_nasc','idade','cd_
mun_res']]
            # adiciona o dataframe à lista de dataframes
            dfs.append(df)
            print(f'{arquivo} concluido indo para o proximo')

    # concatena os dataframes em um único dataframe final
    df_group = pd.concat(dfs, ignore_index=True)
    # Baixa a base de população por município
    #Adicionar Coluna Para os quadrimestres
     df_group.rename(columns={"ano_obito": "ano", "quad_obito":
"quad", 'idade': "idade_dias"}, inplace=True)
    # Merge os dataframes com base nas condições especificadas
    #df_group.drop(['ano'], axis=1, inplace=True)
    df_group.info()

    return df_group


def quadrim_end_date(year: int, q: int) -> str:
    # Q1 → abr, Q2 → ago, Q3 → dez
    end_month = {1: 4, 2: 8, 3: 12}[q]
    last_day = monthrange(year, end_month)[1]
    return date(year, end_month, last_day).isoformat()



def processa_df_sisab(path, data_folder):
    path = data_folder + "/" + path
    df = pd.read_csv(path, encoding='latin-1',
                           skiprows=8,
                           skipfooter=2,
                           engine="python",
                           sep=";")

    # colunas fixas
    if "Unnamed: 13" in df.columns:
        df = df.drop(columns=["Unnamed: 13"])
        fixas = ["UF", "IBGE", "Munícipio"]

    # todas as outras viram períodos
    periodo_cols = [c for c in df.columns if c not in fixas]
```

```
    # derrete
    df_long = df.melt(
      id_vars=fixas,
      value_vars=periodo_cols,
      var_name="periodo_raw",
      value_name="porcentagem"
    )

    # extrai ano e quadrimestre
                                                tmp            =
df_long["periodo_raw"].str.extract(r"(?P<ano>\d{4})\s+Q(?P<q>[
1-3])")
    df_long["ano"] = tmp["ano"].astype(int)
    df_long["q"] = tmp["q"].astype(int)

    # cria coluna data
      df_long["data"] = [quadrim_end_date(y, q) for y, q in
zip(df_long["ano"], df_long["q"])]
          df_long["data"]   =   pd.to_datetime(df_long["data"],
errors="coerce")

    # mantém só as colunas desejadas
      df_final = df_long[["UF", "IBGE", "Munícipio", "data",
"porcentagem"]]
    return df_final
```

APÊNDICE B - ANÁLISE EXPLORATÓRIA DOS DADOS

```
# -*- coding: utf-8 -*-
# Setup
"""

from google.cloud import bigquery
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from google.colab import drive
import os
import numpy as np
from matplotlib.patches import Patch
from datetime import datetime
from matplotlib.ticker import PercentFormatter
from matplotlib.lines import Line2D
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller

# Cria o cliente BigQuery (precisa estar autenticado com
GOOGLE_APPLICATION_CREDENTIALS)
# Monta o Google Drive
drive.mount('/content/drive')

# Defina a variável de ambiente com o caminho do arquivo JSON
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]                    =
"/content/drive/MyDrive/Colab
Notebooks/TCC/tcc-pedro-461615-6bc2e4765d70.json"

client = bigquery.Client()


# Sua query Nascidos Vivos
query = """
SELECT *
FROM `tcc-pedro-461615.mortalidade_infantil.nascidos_vivos`
"""
# Executa a query e transforma em DataFrame
nascidos_vivos = client.query(query).to_dataframe()

# Sua query Citopatologico
query = """
SELECT *
FROM `tcc-pedro-461615.dados_sisab.citopatologico_csv`
"""
# Executa a query e transforma em DataFrame
citopatologico = client.query(query).to_dataframe()

# Sua query Gestantes
query = """
SELECT *
FROM `tcc-pedro-461615.dados_sisab.gestantes_6_consultas_csv`
```

```
"""
# Executa a query e transforma em DataFrame
gestantes_consultas = client.query(query).to_dataframe()

# Sua query Gestantes Consultas Odontologias
query = """
SELECT *
FROM `tcc-pedro-461615.dados_sisab.gestantes_odonto_csv`
"""
# Executa a query e transforma em DataFrame
gestantes_odonto = client.query(query).to_dataframe()

# Sua query Gestantes Sifilis HIV
query = """
SELECT *
FROM `tcc-pedro-461615.dados_sisab.gestantes_sifilis_hiv_csv`
"""
# Executa a query e transforma em DataFrame
gestantes_sifilis = client.query(query).to_dataframe()

# Sua query Mortalidade
query = """
SELECT *
FROM
`tcc-pedro-461615.mortalidade_infantil.mortalidade_infantil`
"""
# Executa a query e transforma em DataFrame
mortalidade = client.query(query).to_dataframe()

def transformar_datas(df, col_data="data"):
    """
     Recebe um DataFrame com uma coluna de datas e retorna o
mesmo DataFrame
    com duas novas colunas: ano e quadrimestre.
    A coluna de data original é removida.
    """
    # Garante que a coluna está em formato datetime
    df[col_data] = pd.to_datetime(df[col_data])

    # Cria as colunas ano e quadrimestre
    df["ano"] = df[col_data].dt.year
    df["quad"] = (df[col_data].dt.month - 1) // 3

    # Remove a coluna original de data
    df = df.drop(columns=[col_data])

    return df

"""# Transformacao Dados

"""
```

```
municipios = citopatologico[['IBGE', 'UF',
'Munícipio']].drop_duplicates()
municipios.rename(columns={'Munícipio': 'municipio'},
inplace=True)
municipios.head()

nascidos_vivos["cd_mun_res"] =
nascidos_vivos["cd_mun_res"].astype(int)
nascidos_vivos["ano"] = nascidos_vivos["ano"].astype(int)
nascidos_vivos.head()

citopatologico = transformar_datas(citopatologico)
citopatologico.drop(columns=['UF', 'Munícipio'], inplace=True)
citopatologico.rename(columns={'porcentagem':
'porcentagem_citopatologico', 'IBGE': 'cd_mun_res'},
inplace=True)
citopatologico.head()

gestantes_consultas = transformar_datas(gestantes_consultas)
gestantes_consultas.drop(columns=['UF', 'Munícipio'],
inplace=True)
gestantes_consultas.rename(columns={'porcentagem':
'porcentagem_consultas_gestantes', 'IBGE': 'cd_mun_res'},
inplace=True)
gestantes_consultas.head()

gestantes_odonto = transformar_datas(gestantes_odonto)
gestantes_odonto.drop(columns=['UF', 'Munícipio'],
inplace=True)
gestantes_odonto.rename(columns={'porcentagem':
'porcentagem_gestantes_odonto', 'IBGE': 'cd_mun_res'},
inplace=True)
gestantes_odonto.head()

gestantes_sifilis = transformar_datas(gestantes_sifilis)
gestantes_sifilis.drop(columns=['UF', 'Munícipio'],
inplace=True)
gestantes_sifilis.rename(columns={'porcentagem':
'porcentagem_gestantes_sifilis', 'IBGE': 'cd_mun_res'},
inplace=True)
gestantes_sifilis.head()

mortalidade.drop(columns=['dt_obito', 'dt_nasc',
'idade_dias'], inplace=True)
mortalidade  = (
    mortalidade
    .groupby(["cd_mun_res", "ano", "quad"])
    .size()
    .reset_index(name="numero_obitos")
)
```

```
mortalidade["cd_mun_res"]                                                =
mortalidade["cd_mun_res"].astype(int)
mortalidade["ano"] = mortalidade["ano"].astype(int)
mortalidade.head()

# Começa com nascidos_vivos
df_final = nascidos_vivos

# Junta mortalidade
df_final   =   df_final.merge(mortalidade,   on=["cd_mun_res",
"ano", "quad"], how="outer")

# Junta gestantes (6 consultas)
df_final         =         df_final.merge(gestantes_consultas,
on=["cd_mun_res", "ano", "quad"], how="outer")

# Junta gestantes odontologia
df_final  =  df_final.merge(gestantes_odonto,  on=["cd_mun_res",
"ano", "quad"], how="outer")

# Junta gestantes sífilis HIV
df_final  =  df_final.merge(gestantes_sifilis,  on=["cd_mun_res",
"ano", "quad"], how="outer")

# Junta citopatológico
df_final   =   df_final.merge(citopatologico,   on=["cd_mun_res",
"ano", "quad"], how="outer")

df_final["taxa_mortalidade"] = (
     df_final["numero_obitos"]  /  df_final["total_nascidos"]  *
1000
)

cols_check = [
    "porcentagem_consultas_gestantes",
    "porcentagem_gestantes_odonto",
    "porcentagem_gestantes_sifilis",
    "porcentagem_citopatologico"
]

df_final = df_final.dropna(subset=cols_check, how="all")
df_filtrado     =     df_final[df_final["numero_obitos"]     <
df_final["total_nascidos"]].copy()
df_final.drop(columns=['total_nascidos',      'numero_obitos'],
inplace=True)
df = df_final

df_final

"""# Analise de Dados
```

```
Checar se há estacionariedade nas séries (tendência/sazonalidade marcantes).

Criar indicadores derivados:

Crescimento percentual ano a ano de nascidos/óbitos.

Taxa de cobertura (consultas, odontologia, sífilis, citopatológico).

Defasagens (lag) para capturar dependência temporal.
"""

def contar_nulos(df):
    nulos_abs = df.isna().sum()
       nulos_pct = (nulos_abs / len(df) * 100).round(2) if len(df) > 0 else 0
                return pd.concat([nulos_abs.rename("nulos"), nulos_pct.rename("pct_nulos")], axis=1)

# Retirada dos Dados de 2025 -> Sem dados de mortalidade
df_num = df[(df['ano'] < 2025)]

#Retirada de Nulos Maior Que 80%
df_num = df_num.assign(taxa_mortalidade_nulo = df_num.groupby("cd_mun_res")["taxa_mortalidade"].transform(lambda x: x.isna().sum()))

df = df_num[(df_num['taxa_mortalidade_nulo'] < 7)]
#  --- 2) Estatísticas descritivas: média, mediana, desvio padrão, min, max ---
# (resultado com cada métrica em colunas; linhas = variáveis)
df_stats = df_num.agg(["mean", "median", "std", "min", "max"]).T
df_stats = df_stats[["mean", "median", "std", "min", "max"]]
# ordena as colunas

# --- 3) Nulos por coluna (em todo o DF, não só nas numéricas) ---
nulos_abs = df_num.isna().sum()
nulos_pct = (nulos_abs / len(df_num) * 100).round(2)
df_nulos = pd.concat([nulos_abs.rename("nulos"), nulos_pct.rename("pct_nulos")], axis=1) \
                      .sort_values(by=["nulos", "pct_nulos"], ascending=False)

print("\n=== Media,Mediana, Desvio Padrao, Minimo e Maximo ===")
print(df_stats)

print("\n=== Nulos Totais ===")
```

```
print(df_nulos)

df['ano'].unique()

df_plot = df.copy()
df_plot["periodo"]  =  df_plot["ano"].astype(str)  +  "-Q"  +
df_plot["quad"].astype(str)

# --- ordena pelo tempo ---
df_plot = df_plot.sort_values(by=["ano", "quad"])

# --- agrupa e calcula média da taxa de mortalidade por
período ---
df_media       =       df_plot.groupby(["ano",       "quad"],
as_index=False)["taxa_mortalidade"].mean()
df_media["periodo"]  =  df_media["ano"].astype(str)  +  "-Q"  +
df_media["quad"].astype(str)

# --- gráfico de linha ---
plt.figure(figsize=(10,5))
plt.plot(df_media["periodo"],     df_media["taxa_mortalidade"],
marker="o", linestyle="-")

# --- ajustes visuais ---
plt.title("Evolução da Taxa de Mortalidade por Quadrimestre",
fontsize=14)
plt.xlabel("Período (Ano-Quadrimestre)", fontsize=12)
plt.ylabel("Taxa de Mortalidade", fontsize=12)
plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.6)

plt.tight_layout()
plt.show()

# --- Quartis e IQR da taxa_mortalidade ---
Q1 = df["taxa_mortalidade"].quantile(0.25)
Q3 = df["taxa_mortalidade"].quantile(0.75)
IQR = Q3 - Q1

limite_inferior = Q1 - 1.5 * IQR
limite_superior = Q3 + 1.5 * IQR

print("Q1:", Q1)
print("Q3:", Q3)
print("IQR:", IQR)
print("Limite inferior:", limite_inferior)
print("Limite superior:", limite_superior)

# --- Filtrar outliers ---
outliers = df[(df["taxa_mortalidade"] < limite_inferior) |
              (df["taxa_mortalidade"] > limite_superior)]
```

```
df = df[(df["taxa_mortalidade"] > limite_inferior) &
              (df["taxa_mortalidade"] < limite_superior)]

print(f"Total de outliers encontrados: {len(outliers)}")

# Mostrar os top 10 outliers
print(outliers.sort_values("taxa_mortalidade",
ascending=False).tail(10))

df_plot = df.copy()
df_plot["periodo"]  =  df_plot["ano"].astype(str)  +  "-Q"  +
df_plot["quad"].astype(str)

# --- ordena pelo tempo ---
df_plot = df_plot.sort_values(by=["ano", "quad"])

#  ---  agrupa  e  calcula  média  da  taxa  de  mortalidade  por
período ---
df_media       =       df_plot.groupby(["ano",       "quad"],
as_index=False)["taxa_mortalidade"].mean()
df_media["periodo"]  =  df_media["ano"].astype(str)  +  "-Q"  +
df_media["quad"].astype(str)

# --- gráfico de linha ---
plt.figure(figsize=(10,5))
plt.plot(df_media["periodo"],     df_media["taxa_mortalidade"],
marker="o", linestyle="-")

# --- ajustes visuais ---
plt.title("Evolução da Taxa de Mortalidade por Quadrimestre",
fontsize=14)
plt.xlabel("Período (Ano-Quadrimestre)", fontsize=12)
plt.ylabel("Taxa de Mortalidade", fontsize=12)
plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.6)

plt.tight_layout()
plt.show()

plt.figure(figsize=(10,6))
sns.boxplot(data=df, x="quad", y="taxa_mortalidade")

plt.title("Boxplot da Taxa de Mortalidade por Quadrimestre")
plt.xlabel("Quadrimestre")
plt.ylabel("Taxa de Mortalidade")
plt.show()

# Seleciona apenas as variáveis quantitativas de interesse
cols = [
    "taxa_mortalidade",
```

```
    "porcentagem_consultas_gestantes",
    "porcentagem_gestantes_odonto",
    "porcentagem_gestantes_sifilis",
    "porcentagem_citopatologico"
]

plt.figure(figsize=(8,6))
sns.heatmap(df[cols].corr(),    annot=True,    cmap="coolwarm",
fmt=".2f")
plt.title("Heatmap de Correlação entre Variáveis")
plt.show()

# Agrega e ordena
serie_df = (
                                df.groupby(["ano",        "quad"],
as_index=False)["taxa_mortalidade"]
      .mean()
      .sort_values(["ano", "quad"])
)

#  Mapeia  quadrimestres  para  meses  centrais:  1→Fev,  2→Jun,
3→Out (ajuste se preferir)
map_mes = {1: 2, 2: 6, 3: 10}
serie_df["mes"] = serie_df["quad"].map(map_mes)

# Cria datetime
serie_df["data"]   =   pd.to_datetime(dict(year=serie_df["ano"],
month=serie_df["mes"], day=1))

y = serie_df["taxa_mortalidade"].values
x = serie_df["data"].values

#plt.figure(figsize=(12,5))
#plt.plot(x, y, marker="o")
#plt.title("Taxa de Mortalidade ao longo do tempo")
#plt.xlabel("Tempo")
#plt.ylabel("Taxa de Mortalidade")
#plt.show()

#  Decomposição  e  ADF  (periodicidade  =  3,  já  que  são  3
quadrimestres/ano)
serie = pd.Series(y, index=serie_df["data"])
decomp   =   sm.tsa.seasonal_decompose(serie,   model="additive",
period=3)
decomp.plot()
plt.show()

adf = adfuller(serie.dropna())
print("ADF:", adf[0], "p-value:", adf[1])

# Agrega e ordena
```

```
serie_df = (
                                df.groupby(["ano",       "quad"],
as_index=False)["taxa_mortalidade"]
      .mean()
      .sort_values(["ano", "quad"])
)

# Mapeia quadrimestres para meses centrais: 1→Fev, 2→Jun,
3→Out (ajuste se preferir)
map_mes = {1: 2, 2: 6, 3: 10}
serie_df["mes"] = serie_df["quad"].map(map_mes)

# Cria datetime
serie_df["data"]  =  pd.to_datetime(dict(year=serie_df["ano"],
month=serie_df["mes"], day=1))

y = serie_df["taxa_mortalidade"].values
x = serie_df["data"].values

#plt.figure(figsize=(12,5))
#plt.plot(x, y, marker="o")
#plt.title("Taxa de Mortalidade ao longo do tempo")
#plt.xlabel("Tempo")
#plt.ylabel("Taxa de Mortalidade")
#plt.show()

# Decomposição e ADF (periodicidade = 3, já que são 3
quadrimestres/ano)
serie = pd.Series(y, index=serie_df["data"])
decomp            =            sm.tsa.seasonal_decompose(serie,
model="multiplicative", period=3)
decomp.plot()
plt.show()

adf = adfuller(serie.dropna())
print("ADF:", adf[0], "p-value:", adf[1])

from statsmodels.tsa.stattools import adfuller

resid = decomp.resid.dropna()
adf = adfuller(resid)
print("ADF:", adf[0], "p-value:", adf[1])

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plot_acf(resid.dropna(), lags=3)
plot_pacf(resid.dropna(), lags=3)
plt.show()
```

APÊNDICE C - OTIMIZAÇÃO E TREINAMENTO DO MODELO

```
# -*- coding: utf-8 -*-
"""1.4  Resultados TCC
# Setup
"""

!pip install scikit-optimize
!pip install pmdarima
import os
import math
from datetime import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import Patch
from matplotlib.ticker import PercentFormatter
from matplotlib.lines import Line2D
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.statespace.sarimax import SARIMAX
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from  sklearn.metrics  import  mean_squared_error,  r2_score,
mean_absolute_percentage_error
from sklearn.model_selection import TimeSeriesSplit
from skopt import BayesSearchCV
from skopt.space import Real, Integer, Categorical
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from google.cloud import bigquery
from google.colab import drive

#  Cria  o  cliente  BigQuery  (precisa  estar  autenticado  com
GOOGLE_APPLICATION_CREDENTIALS)
# Monta o Google Drive
drive.mount('/content/drive')

# Defina a variável de ambiente com o caminho do arquivo JSON
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]                    =
"/content/drive/MyDrive/Colab
Notebooks/TCC/tcc-pedro-461615-6bc2e4765d70.json"

client = bigquery.Client()


# Sua query Nascidos Vivos
query = """
```

```
SELECT *
FROM `tcc-pedro-461615.mortalidade_infantil.nascidos_vivos`
"""
# Executa a query e transforma em DataFrame
nascidos_vivos = client.query(query).to_dataframe()

# Sua query Citopatologico
query = """
SELECT *
FROM `tcc-pedro-461615.dados_sisab.citopatologico_csv`
"""
# Executa a query e transforma em DataFrame
citopatologico = client.query(query).to_dataframe()

# Sua query Gestantes
query = """
SELECT *
FROM `tcc-pedro-461615.dados_sisab.gestantes_6_consultas_csv`
"""
# Executa a query e transforma em DataFrame
gestantes_consultas = client.query(query).to_dataframe()

# Sua query Gestantes Consultas Odontologias
query = """
SELECT *
FROM `tcc-pedro-461615.dados_sisab.gestantes_odonto_csv`
"""
# Executa a query e transforma em DataFrame
gestantes_odonto = client.query(query).to_dataframe()

# Sua query Gestantes Sifilis HIV
query = """
SELECT *
FROM `tcc-pedro-461615.dados_sisab.gestantes_sifilis_hiv_csv`
"""
# Executa a query e transforma em DataFrame
gestantes_sifilis = client.query(query).to_dataframe()

# Sua query Mortalidade
query = """
SELECT *
FROM
`tcc-pedro-461615.mortalidade_infantil.mortalidade_infantil`
"""
# Executa a query e transforma em DataFrame
mortalidade = client.query(query).to_dataframe()

query = """
SELECT
    dados.id_municipio AS id_municipio,
    diretorio_id_municipio.nome AS id_municipio_nome,
```

```
    dados.populacao as populacao
FROM
`basedosdados.br_ibge_censo_2022.alfabetizacao_grupo_idade_sexo_raca` AS dados
LEFT JOIN (SELECT DISTINCT id_municipio,nome FROM `basedosdados.br_bd_diretorios_brasil.municipio`) AS diretorio_id_municipio
                            ON dados.id_municipio =
diretorio_id_municipio.id_municipio
"""
# Executa a query e transforma em DataFrame
populacao = client.query(query).to_dataframe()

def transformar_datas(df, col_data="data"):
    """
     Recebe um DataFrame com uma coluna de datas e retorna o mesmo DataFrame
    com duas novas colunas: ano e quadrimestre.
    A coluna de data original é removida.
    """
    # Garante que a coluna está em formato datetime
    df[col_data] = pd.to_datetime(df[col_data])

    # Cria as colunas ano e quadrimestre
    df["ano"] = df[col_data].dt.year
    df["quad"] = (df[col_data].dt.month - 1) // 3

    # Remove a coluna original de data
    df = df.drop(columns=[col_data])

    return df

"""# Transformacao Dados

"""

populacao.rename(columns={'id_municipio': 'cd_mun_res'}, inplace=True)
populacao['cd_mun_res'] =
populacao['cd_mun_res'].astype(str).str.slice(stop=6)
populacao.dropna(inplace=True)
populacao = (
    populacao.groupby('cd_mun_res', as_index=False)
    .agg({
        'populacao': 'sum',
        'id_municipio_nome': 'first'
    })
)
populacao.head()

populacao
```

```
municipios = citopatologico[['IBGE', 'UF',
'Munícipio']].drop_duplicates()
municipios.rename(columns={'Munícipio': 'municipio', 'IBGE':
'cd_mun_res'}, inplace=True)
municipios["cd_mun_res"] =
municipios["cd_mun_res"].astype(str)
municipios.head()

nascidos_vivos["cd_mun_res"] =
nascidos_vivos["cd_mun_res"].astype(str)
nascidos_vivos["ano"] = nascidos_vivos["ano"].astype(int)
nascidos_vivos.head()

citopatologico = transformar_datas(citopatologico)
citopatologico.drop(columns=['UF', 'Munícipio'], inplace=True)
citopatologico.rename(columns={'porcentagem':
'porcentagem_citopatologico', 'IBGE': 'cd_mun_res'},
inplace=True)
citopatologico["cd_mun_res"] =
citopatologico["cd_mun_res"].astype(str)
citopatologico.head()

gestantes_consultas = transformar_datas(gestantes_consultas)
gestantes_consultas.drop(columns=['UF', 'Munícipio'],
inplace=True)
gestantes_consultas.rename(columns={'porcentagem':
'porcentagem_consultas_gestantes', 'IBGE': 'cd_mun_res'},
inplace=True)
gestantes_consultas["cd_mun_res"] =
gestantes_consultas["cd_mun_res"].astype(str)
gestantes_consultas.head()

gestantes_odonto = transformar_datas(gestantes_odonto)
gestantes_odonto.drop(columns=['UF', 'Munícipio'],
inplace=True)
gestantes_odonto.rename(columns={'porcentagem':
'porcentagem_gestantes_odonto', 'IBGE': 'cd_mun_res'},
inplace=True)
gestantes_odonto["cd_mun_res"] =
gestantes_odonto["cd_mun_res"].astype(str)
gestantes_odonto.head()

gestantes_sifilis = transformar_datas(gestantes_sifilis)
gestantes_sifilis.drop(columns=['UF', 'Munícipio'],
inplace=True)
gestantes_sifilis.rename(columns={'porcentagem':
'porcentagem_gestantes_sifilis', 'IBGE': 'cd_mun_res'},
inplace=True)
gestantes_sifilis["cd_mun_res"] =
gestantes_sifilis["cd_mun_res"].astype(str)
```

```
gestantes_sifilis.head()

mortalidade.drop(columns=['dt_obito',                'dt_nasc',
'idade_dias'], inplace=True)
mortalidade  = (
    mortalidade
    .groupby(["cd_mun_res", "ano", "quad"])
    .size()
    .reset_index(name="numero_obitos")
)
mortalidade["cd_mun_res"]                                    =
mortalidade["cd_mun_res"].astype(str)
mortalidade["ano"] = mortalidade["ano"].astype(int)
mortalidade.head()

def contar_nulos(df):
    nulos_abs = df.isna().sum()
       nulos_pct  =  (nulos_abs  /  len(df)  *  100).round(2)  if
len(df) > 0 else 0
                return    pd.concat([nulos_abs.rename("nulos"),
nulos_pct.rename("pct_nulos")], axis=1)

"""# Gerar DF"""

# Começa com nascidos_vivos
df = nascidos_vivos

# Junta mortalidade
df  =  df.merge(mortalidade,  on=["cd_mun_res",  "ano",  "quad"],
how="outer")

# Junta gestantes (6 consultas)
df  =  df.merge(gestantes_consultas,  on=["cd_mun_res",  "ano",
"quad"], how="outer")

# Junta gestantes odontologia
df   =   df.merge(gestantes_odonto,   on=["cd_mun_res",   "ano",
"quad"], how="outer")

# Junta gestantes sífilis HIV
df   =   df.merge(gestantes_sifilis,   on=["cd_mun_res",   "ano",
"quad"], how="outer")

# Junta citopatológico
df   =   df.merge(citopatologico,   on=["cd_mun_res",   "ano",
"quad"], how="outer")

df["taxa_mortalidade"] = (
    df["numero_obitos"] / df["total_nascidos"] * 1000
)
```

```
cols_check = [
    "porcentagem_consultas_gestantes",
    "porcentagem_gestantes_odonto",
    "porcentagem_gestantes_sifilis",
    "porcentagem_citopatologico"
]

df = df.dropna(subset=cols_check, how="all")
df = df[df["numero_obitos"] < df["total_nascidos"]].copy()


df    =    df.merge(populacao[['cd_mun_res',    'populacao']],
on=["cd_mun_res"], how="left")

df = df.merge(municipios[['cd_mun_res', 'UF']], how="left")
#ufs_sul_sudeste = ['RS', 'SC', 'PR', 'SP', 'RJ', 'MG', 'ES']
#df = df[df['UF'].isin(ufs_sul_sudeste)]

# Retirada dos Dados de 2025 -> Sem dados de mortalidade
df  =  df[(df['populacao']  >  100000)  &  (df['populacao']  <
900000)]
df_completo = df.copy()
df = df[(df['ano'] < 2025)]

#Retirada de Nulos Maior Que 80%
df         =         df.assign(taxa_mortalidade_nulo         =
df.groupby("cd_mun_res")["taxa_mortalidade"].transform(lambda
x: x.isna().sum()))

df = df[(df['taxa_mortalidade_nulo'] < 7)]

# --- Quartis e IQR da taxa_mortalidade ---
p1 = df["taxa_mortalidade"].quantile(0.10)
Q1 = df["taxa_mortalidade"].quantile(0.25)
Q3 = df["taxa_mortalidade"].quantile(0.75)
IQR = Q3 - Q1
limite_superior = Q3 + 1 * IQR


df = df[(df["taxa_mortalidade"] > p1) &
              (df["taxa_mortalidade"] < limite_superior)]


df.drop(columns=['total_nascidos',             'numero_obitos',
"taxa_mortalidade_nulo", "cd_mun_res"], inplace=True)
df_completo.drop(columns=['total_nascidos',    'numero_obitos',
"cd_mun_res"], inplace=True)
df_raw = df.copy()

plt.figure(figsize=(6, 5))
```

```
# Agrupa por quadrimestre e cria uma lista de listas com as
taxas
data = [group["taxa_mortalidade"].values for name, group in
df.groupby("quad")]

# Cria o boxplot
plt.boxplot(
    data,
    vert=True,
    patch_artist=True,
    boxprops=dict(facecolor="skyblue", color="black"),
    medianprops=dict(color="red", linewidth=2),
    whiskerprops=dict(color="black"),
    capprops=dict(color="black"),
)

# Define rótulos e título
#plt.title("Boxplot - Taxa de Mortalidade por Quadrimestre")
plt.xlabel("Quadrimestre")
plt.ylabel("Taxa de Mortalidade")
plt.xticks(range(1,          len(data)          +          1),
sorted(df["quad"].unique()))
plt.grid(axis="y", linestyle="--", alpha=0.4)
plt.tight_layout()
plt.show()

df_plot = df.copy()
df_plot["periodo"]  =  df_plot["ano"].astype(str)  +  "-Q"  +
df_plot["quad"].astype(str)

# --- ordena pelo tempo ---
df_plot = df_plot.sort_values(by=["ano", "quad"])

# --- agrupa e calcula média da taxa de mortalidade por
período ---
df_media       =       df_plot.groupby(["ano",       "quad"],
as_index=False)["taxa_mortalidade"].mean()
df_media["periodo"]  =  df_media["ano"].astype(str)  +  "-Q"  +
df_media["quad"].astype(str)

# --- gráfico de linha ---
plt.figure(figsize=(10,5))
plt.plot(df_media["periodo"],     df_media["taxa_mortalidade"],
marker="o", linestyle="-")

# --- ajustes visuais ---
plt.title("Evolução da Taxa de Mortalidade por Quadrimestre em
Cidades de Grande Porte", fontsize=14)
plt.xlabel("Período (Ano-Quadrimestre)", fontsize=12)
plt.ylabel("Taxa de Mortalidade", fontsize=12)
plt.xticks(rotation=45)
```

```
plt.grid(True, linestyle="--", alpha=0.6)

plt.tight_layout()
plt.show()

# Retirada dos Dados de 2025 -> Sem dados de mortalidade
df_num = df[(df['ano'] < 2025)].copy()

#Retirada de Nulos Maior Que 80%

#df = df_num[(df_num['taxa_mortalidade_nulo'] < 7)]

#  --- 2) Estatísticas descritivas: média, mediana, desvio
padrão, min, max ---
df_stats = df_num[
    [
        "porcentagem_consultas_gestantes",
        "porcentagem_gestantes_odonto",
        "porcentagem_gestantes_sifilis",
        "porcentagem_citopatologico",
        "taxa_mortalidade"
    ]
].agg(["mean", "median", "std", "min", "max"]).T

# (resultado com cada métrica em colunas; linhas = variáveis)
df_stats = df_stats[["mean", "median", "std", "min", "max"]]
# ordena as colunas

# --- 3) Nulos por coluna (em todo o DF, não só nas numéricas)
---
nulos_abs = df_num.isna().sum()
nulos_pct = (nulos_abs / len(df_num) * 100).round(2)
df_nulos        =        pd.concat([nulos_abs.rename("nulos"),
nulos_pct.rename("pct_nulos")], axis=1) \
                        .sort_values(by=["nulos", "pct_nulos"],
ascending=False)

print("\n===  Media,Mediana,  Desvio  Padrao,  Minimo  e  Maximo
===")
print(df_stats)

"""# Modelo Otimizado



"""

df = df_raw.copy()
df = pd.get_dummies(df, columns=['UF'], prefix='UF')

X = df.drop("taxa_mortalidade", axis=1)
```

```
y = df["taxa_mortalidade"]

cond_treino = (X["ano"] <= 2023) | ((X["ano"] == 2024) &
(X["quad"] == 1))

# separa conjuntos
X_train = X[cond_treino]
y_train = y[cond_treino]

X_test = X[~cond_treino]
y_test = y[~cond_treino]

def add_quad_trig(df_in):
    out = df_in.copy()
    out["quad"] = out["quad"].astype(int)
    # quadrimestre = 3 ciclos/ano -> divisor = 3
    out["quad_sin"] = np.sin(2 * np.pi * out["quad"] / 3)
    out["quad_cos"] = np.cos(2 * np.pi * out["quad"] / 3)
    out = out.drop(columns=["quad"])
    return out

X_train = add_quad_trig(X_train)
X_test  = add_quad_trig(X_test)

df_completo_ml = df_completo.copy()
df_completo_ml         =         pd.get_dummies(df_completo_ml,
columns=['UF'], prefix='UF')

X_completo = df_completo_ml.drop("taxa_mortalidade", axis=1)

y_completo = df_completo_ml["taxa_mortalidade"]

cond_treino = (X_completo["ano"] <= 2024)

# separa conjuntos
X_train_completo = X_completo[cond_treino]
y_train_completo = y_completo[cond_treino]

X_train_completo = add_quad_trig(X_train_completo)

# Criar modelo (regressão porque o target é numérico)
model = XGBRegressor(
    colsample_bylevel=0.3,
    colsample_bytree=1.0,
    gamma=2.3526454841602398,
    learning_rate=0.001,
    max_bin=512,
    max_depth=30,
    min_child_weight=29,
    n_estimators=4798,
```

```
    reg_alpha=8.512582403794102,
    reg_lambda=100.0,
    subsample=1.0
)

# Treinar
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Avaliação
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print("MSE:", mse)
print("RMSE:", rmse)
print("R²:", r2)
print("MAPE:", mape)

# garante que os dados estão alinhados por índice
y_test = pd.Series(y_test).reset_index(drop=True)
y_pred = pd.Series(y_pred).reset_index(drop=True)

plt.figure(figsize=(9, 5))
plt.plot(y_test, label="Real", marker="o", linestyle="-")
plt.plot(y_pred, label="Previsto", marker="x", linestyle="--")

plt.xlabel("Observações (tempo)")
plt.ylabel("Taxa de Mortalidade (original)")
plt.title("Regressão Linear Múltipla — Real vs Previsto
(desnormalizado)")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)
plt.tight_layout()
plt.show()

# --- 1) Junta as bases ---
df_previsto = X_test.copy()
df_previsto["y_real"] = y_test.values
df_previsto["y_pred"] = y_pred

# --- 2) Garante os tipos ---
df_previsto["ano"] = df_previsto["ano"].astype(int)
df_previsto["quad"] = df_previsto["quad"].astype(int)

# --- 3) Agrupa por ano e quadrimestre (média das taxas) ---
df_media = (
                        df_previsto.groupby(["ano",      "quad"],
as_index=False)[["y_real", "y_pred"]]
```

```
    .mean()
    .sort_values(["ano", "quad"])
)
df_media["periodo"] = df_media["ano"].astype(str) + "-Q" +
df_media["quad"].astype(str)

# --- 4) Gráfico de linha ---
plt.figure(figsize=(10, 5))
plt.plot(df_media["periodo"], df_media["y_real"], marker="o",
label="Real")
plt.plot(df_media["periodo"], df_media["y_pred"], marker="o",
label="Previsto")

plt.title("Série Temporal — Mortalidade Real vs Prevista",
fontsize=14)
plt.xlabel("Período (Ano-Quadrimestre)", fontsize=12)
plt.ylabel("Taxa de Mortalidade", fontsize=12)
plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.6)
plt.legend()
plt.tight_layout()
plt.show()

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# --- 1) Reconstrói df_previsto com ano, quad, real e previsto
---
df_previsto = X_test.copy()
df_previsto["y_real"] = y_test.values
df_previsto["y_pred"] = y_pred

# --- 2) Recria a coluna 'quad' a partir de seno e cosseno ---
angles = np.arctan2(df_previsto["quad_sin"],
df_previsto["quad_cos"])
df_previsto["quad"] = ((np.round((angles % (2*np.pi)) /
(2*np.pi/3))) % 3).astype(int)
df_previsto["quad"].replace(0, 3, inplace=True)

# --- 3) Garante tipos ---
df_previsto["ano"] = df_previsto["ano"].astype(int)
df_previsto["quad"] = df_previsto["quad"].astype(int)

# --- 4) Agrupa médias do conjunto previsto ---
df_media_prev = (
                    df_previsto.groupby(["ano", "quad"],
as_index=False)[["y_real", "y_pred"]]
    .mean()
    .sort_values(["ano", "quad"])
)
```

```
df_media_prev["periodo"] = df_media_prev["ano"].astype(str) +
"-Q" + df_media_prev["quad"].astype(str)

# --- 5) Série histórica completa (df_raw) ---
df_raw["ano"] = df_raw["ano"].astype(int)
df_raw["quad"] = df_raw["quad"].astype(int)
df_hist = (
                          df_raw.groupby(["ano",      "quad"],
as_index=False)["taxa_mortalidade"]
    .mean()
    .sort_values(["ano", "quad"])
)
df_hist["periodo"] = df_hist["ano"].astype(str) + "-Q" +
df_hist["quad"].astype(str)

# --- 6) Gráfico com ambas as séries ---
plt.figure(figsize=(10, 5))
plt.plot(df_hist["periodo"],       df_hist["taxa_mortalidade"],
marker="o", label="Real (Histórico)")
plt.plot(df_media_prev["periodo"],     df_media_prev["y_pred"],
marker="o", label="Previsto (Modelo)")

# --- 7) Opcional: marcar início do teste ---
inicio_teste = df_media_prev["periodo"].min()
plt.axvline(inicio_teste,     color="gray",     linestyle="--",
alpha=0.7, label=f"Início Predicao
 ({inicio_teste})")

# --- 8) Estilo do gráfico ---
plt.title("Série Temporal — Mortalidade Real vs Prevista",
fontsize=14)
plt.xlabel("Período (Ano-Quadrimestre)", fontsize=12)
plt.ylabel("Taxa de Mortalidade", fontsize=12)
plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.6)
plt.legend()
plt.tight_layout()
plt.show()

# ============================================
# BASE COMPLETA (inclui 2025) + NOWCAST 2025
# ============================================
# Requisitos já existentes no seu ambiente:
#   -   nascidos_vivos,   mortalidade,   gestantes_consultas,
gestantes_odonto,
#   gestantes_sifilis, citopatologico, populacao, municipios
#  - add_quad_trig(df)         -> mesma função usada no treino
(dropa 'quad', cria sin/cos)
#  - model                          -> já treinado até 2024 (ou
conforme seu pipeline)
```

```
# - df_raw                    -> histórico original (para plotar
o "Real (até 2024)")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# ------------ 1) Montagem da base completa (NÃO exclui 2025)
------------
df = nascidos_vivos.copy()
df  =  df.merge(mortalidade,            on=["cd_mun_res",  "ano",
"quad"], how="outer")
df   =   df.merge(gestantes_consultas,  on=["cd_mun_res",  "ano",
"quad"], how="outer")
df  =  df.merge(gestantes_odonto,       on=["cd_mun_res",  "ano",
"quad"], how="outer")
df  =  df.merge(gestantes_sifilis,      on=["cd_mun_res",  "ano",
"quad"], how="outer")
df  =  df.merge(citopatologico,         on=["cd_mun_res",  "ano",
"quad"], how="outer")

mask_2025 = (df["ano"] == 2025)

#  Para  2025:  força  número  de  óbitos  =  0  quando  vier  NaN;
mantém valores reais se existirem
df.loc[mask_2025,    "numero_obitos"]    =    df.loc[mask_2025,
"numero_obitos"].fillna(0)

#  Taxa:  0  para  2025;  demais  anos  calculada  normalmente
(tratando divisão por zero)
df["taxa_mortalidade"] = np.where(
    mask_2025,
    0.0,
    np.where(
        (df["total_nascidos"].fillna(0) > 0),
        (df["numero_obitos"] / df["total_nascidos"]) * 1000,
        np.nan
    )
)

# Filtros que NÃO derrubam 2025
cols_check = [
    "porcentagem_consultas_gestantes",
    "porcentagem_gestantes_odonto",
    "porcentagem_gestantes_sifilis",
    "porcentagem_citopatologico"
]
df   =   df[   mask_2025   |   df[cols_check].notna().any(axis=1)
].copy()
df    =    df[    mask_2025    |    (df["numero_obitos"]    <
df["total_nascidos"]) ].copy()
```

```
# Complementos
df    =    df.merge(populacao[['cd_mun_res',    'populacao']],
on="cd_mun_res", how="left")
df      =      df.merge(municipios[['cd_mun_res',      'UF']],
on="cd_mun_res", how="left")

#  ------------  2)  Dummies  +  features  iguais  ao  treino
------------
df_ml = pd.get_dummies(df, columns=['UF'], prefix='UF')

# Se preferir treinar/avaliar novamente, você faz como no seu
script original.
#  Aqui  seguimos  direto  para  preparar  X_2025  com  as  MESMAS
transformações do treino.
X_full = df_ml.drop(columns=["taxa_mortalidade"])

X_2025_raw = X_full[X_full["ano"] == 2025].copy()
ano_2025  = X_2025_raw["ano"].copy()
quad_2025 = X_2025_raw["quad"].copy()

# MESMA engenharia do treino
X_2025_feat = add_quad_trig(X_2025_raw)

# Alinhamento de colunas ao espaço de features do treino
def _align_to_train(X_feat, model, fallback_cols=None):
    cols_ref = getattr(model, "feature_names_in_", None)
    if cols_ref is None:
        if fallback_cols is not None:
            cols_ref = fallback_cols
        else:
                # Tenta pegar de um X_train_feat existente no
ambiente
            try:
                cols_ref = X_train_feat.columns
            except NameError:
                cols_ref = X_feat.columns  # último recurso
    return X_feat.reindex(columns=cols_ref, fill_value=0)

X_2025_feat = _align_to_train(X_2025_feat, model)

# ------------ 3) Previsão 2025 (nowcast) ------------
y_pred_2025 = model.predict(X_2025_feat)

df_nowcast_2025 = pd.DataFrame({
    "ano": ano_2025.values,
    "quad": quad_2025.values,
    "taxa_prevista": y_pred_2025
})
```

```
df_nowcast_2025["periodo"]                                          =
df_nowcast_2025["ano"].astype(str)       +       "-Q"       +
df_nowcast_2025["quad"].astype(int).astype(str)

#  Agregado  por  período  (caso  haja  múltiplas  linhas  por
período)
nowcast_2025 = (
                              df_nowcast_2025.groupby("periodo",
as_index=True)["taxa_prevista"]
    .mean()
    .sort_index()
)

# ------------ 4) Plot: histórico real (até 2024) + previsão
2025 (linha contínua) ------------
df_hist = df_raw.copy()
df_hist["periodo"]  =  df_hist["ano"].astype(str)  +  "-Q"  +
df_hist["quad"].astype(int).astype(str)
y_real_hist_ate_2024 = (
    df_hist[df_hist["ano"] <= 2024]
    .groupby("periodo", as_index=True)["taxa_mortalidade"]
    .mean()
    .sort_index()
)

# Junta o real (até 2024) + previsão (2025) em uma única série
contínua
serie_continua       =       pd.concat([y_real_hist_ate_2024,
nowcast_2025]).sort_index()

plt.figure(figsize=(10,5))

# linha única contínua
plt.plot(serie_continua.index, serie_continua.values,
         marker="o", linestyle="-", label="Taxa de Mortalidade
NeoNatal Real + Prevista")

# destaca a fronteira entre 2024 e 2025 com linha pontilhada
vertical
ultimo_2024 = y_real_hist_ate_2024.index[-1]
plt.axvline(x=ultimo_2024,   color="orange",   linestyle="--",
alpha=0.6, label="Previsao")

#

plt.title("Taxa de Mortalidade Neonatal — Histórico até 2024 e
Nowcasting 2025 Q1")
plt.xlabel("Período (Ano-Quadrimestre)")
plt.ylabel("Taxa de Mortalidade Neonatal")
plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.6)
```

```
plt.legend()
plt.tight_layout()
plt.show()

"""# Modelo Otimizado - Separacao Por regiao



"""

def add_quad_trig(df_in):
      out = df_in.copy()
      out["quad"] = out["quad"].astype(int)
      # quadrimestre = 3 ciclos/ano -> divisor = 3
      out["quad_sin"] = np.sin(2 * np.pi * out["quad"] / 3)
      out["quad_cos"] = np.cos(2 * np.pi * out["quad"] / 3)
      out = out.drop(columns=["quad"])
      return out

df = df_raw.copy()
df = pd.get_dummies(df, columns=['UF'], prefix='UF')
X = df.drop("taxa_mortalidade", axis=1)
y = df["taxa_mortalidade"]
cond_treino = (X["ano"] <= 2023) | ((X["ano"] == 2024) &
(X["quad"] == 1))
    # separa conjuntos
X_train = X[cond_treino]
y_train = y[cond_treino]
X_test = X[~cond_treino]
y_test = y[~cond_treino]
X_train = add_quad_trig(X_train)
X_test  = add_quad_trig(X_test)
    # Criar modelo (regressão porque o target é numérico)
model = XGBRegressor(
        colsample_bylevel=0.3,
        colsample_bytree=1.0,
        gamma=2.3526454841602398,
        learning_rate=0.001,
        max_bin=512,
        max_depth=30,
        min_child_weight=29,
        n_estimators=4798,
        reg_alpha=8.512582403794102,
        reg_lambda=100.0,
        subsample=1.0
    )

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

    # Avaliação
```

```
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

    # garante que os dados estão alinhados por índice
y_test = pd.Series(y_test).reset_index(drop=True)
y_pred = pd.Series(y_pred).reset_index(drop=True)

plt.figure(figsize=(9, 5))
plt.plot(y_test, label="Real", marker="o", linestyle="-")
plt.plot(y_pred, label="Previsto", marker="x", linestyle="--")

plt.xlabel("Observações (tempo)")
plt.ylabel("Taxa de Mortalidade (original)")
plt.title("Regressão Linear Múltipla — Real vs Previsto
(desnormalizado)")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)
plt.tight_layout()
plt.show()
print("MSE:", mse)
print("RMSE:", rmse)
print("R²:", r2)
print("MAPE:", mape)

df = df_raw.copy()
unique_ufs = sorted(df_raw["UF"].dropna().unique())
for UF in unique_ufs:
  df = df_raw.copy()

  df = pd.get_dummies(df, columns=['UF'], prefix='UF')
  df = df[df[f"UF_{UF}"] == 1]
  X = df.drop("taxa_mortalidade", axis=1)

  y = df["taxa_mortalidade"]

   cond_treino = (X["ano"] <= 2023) | ((X["ano"] == 2024) &
(X["quad"] == 1))

  # separa conjuntos
  X_train = X[cond_treino]
  y_train = y[cond_treino]

  X_test = X[~cond_treino]
  y_test = y[~cond_treino]

  X_train = add_quad_trig(X_train)
  X_test  = add_quad_trig(X_test)
  # Criar modelo (regressão porque o target é numérico)
  #model = XGBRegressor(
```

```
  #    colsample_bylevel=0.3,
  #    colsample_bytree=1.0,
  #    gamma=2.3526454841602398,
  #    learning_rate=0.001,
  #    max_bin=512,
  #    max_depth=30,
  #    min_child_weight=29,
  #    n_estimators=4798,
  #    reg_alpha=8.512582403794102,
  #    reg_lambda=100.0,
  #    subsample=1.0
  #)

  #model.fit(X_train, y_train)
  y_pred = model.predict(X_test)

  # Avaliação
  mse = mean_squared_error(y_test, y_pred)
  rmse = math.sqrt(mse)
  r2 = r2_score(y_test, y_pred)
  mape = mean_absolute_percentage_error(y_test, y_pred)

  # garante que os dados estão alinhados por índice
  y_test = pd.Series(y_test).reset_index(drop=True)
  y_pred = pd.Series(y_pred).reset_index(drop=True)

  plt.figure(figsize=(9, 5))
  plt.plot(y_test, label="Real", marker="o", linestyle="-")
         plt.plot(y_pred,    label="Previsto",    marker="x",
linestyle="--")

  plt.xlabel("Observações (tempo)")
  plt.ylabel("Taxa de Mortalidade (original)")
   plt.title(f"Regressão Linear Múltipla — Real vs Previsto
({UF})")
  plt.legend()
  plt.grid(True, linestyle="--", alpha=0.6)
  plt.tight_layout()
  plt.show()
  print("UF:", UF)
  print("MSE:", mse)
  print("RMSE:", rmse)
  print("R²:", r2)
  print("MAPE:", mape)

regioes = {
    "Sul": ["RS", "SC", "PR"],
    "Sudeste": ["SP", "RJ", "MG", "ES"],
    "Centro-Oeste": ["DF", "GO", "MT", "MS"],
      "Nordeste": ["BA", "SE", "AL", "PE", "PB", "RN", "CE",
"PI", "MA"],
```

```
    "Norte": ["TO", "PA", "AP", "AM", "RR", "RO", "AC"],
    "maiores": ["RR", "AP", "AL", "RO", "PI", "PR", "SE"],
    #"Alto (>30%)": ["AP", "RR", "RO", "AL"],
      #"Médio (15-30%)": ["PI", "PR", "SE", "PB", "AC", "ES",
"MG", "RJ", "RS", "CE", "PA", "AM", "DF"],
       #"Baixo (≤15%)": ["BA", "SP", "PE", "MS", "GO", "MA",
"SC", "MT"]
}


df = df_raw.copy()

for regiao, ufs in regioes.items():

    df = df_raw.copy()
    # Filtra o dataframe para a região atual
    df = pd.get_dummies(df, columns=['UF'], prefix='UF')
      mask = df[[f"UF_{uf}" for uf in ufs if f"UF_{uf}" in
df.columns]].any(axis=1)
    df = df[mask].copy()

    X = df.drop("taxa_mortalidade", axis=1)

    y = df["taxa_mortalidade"]

     cond_treino = (X["ano"] <= 2023) | ((X["ano"] == 2024) &
(X["quad"] == 1))

    X_test = X[~cond_treino]
    y_test = y[~cond_treino]
    X_test  = add_quad_trig(X_test)
    # Criar modelo (regressão porque o target é numérico)
    y_pred = model.predict(X_test)

    # Avaliação
    mse = mean_squared_error(y_test, y_pred)
    rmse = math.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    mape = mean_absolute_percentage_error(y_test, y_pred)

    # garante que os dados estão alinhados por índice
    y_test = pd.Series(y_test).reset_index(drop=True)
    y_pred = pd.Series(y_pred).reset_index(drop=True)

    plt.figure(figsize=(9, 5))
    plt.plot(y_test, label="Real", marker="o", linestyle="-")
             plt.plot(y_pred,   label="Previsto",   marker="x",
linestyle="--")

    plt.xlabel("Observações (tempo)")
    plt.ylabel("Taxa de Mortalidade (original)")
```

```
     plt.title("Regressão Linear Múltipla — Real vs Previsto
(desnormalizado)")
    plt.legend()
    plt.grid(True, linestyle="--", alpha=0.6)
    plt.tight_layout()
    plt.show()
    print("UF:", ufs)
    print("MSE:", mse)
    print("RMSE:", rmse)
    print("R²:", r2)
    print("MAPE:", mape)

# Plot da importância das features
plt.figure(figsize=(8,5))
plt.barh(X_test.columns, model.feature_importances_)
plt.xlabel("Importância")
plt.ylabel("Variável")
plt.title("Importância das Features - XGBoost")
plt.show()

# Cria DataFrame com as features e suas importâncias
df_import = pd.DataFrame({
    "Feature": X_test.columns,
    "Importância": model.feature_importances_
})

# Ordena do maior para o menor
df_import       =       df_import.sort_values(by="Importância",
ascending=False)

# Calcula % acumulada
df_import["%         Acumulado"]         =         100         *
df_import["Importância"].cumsum()                                /
df_import["Importância"].sum()

# Plot
fig, ax1 = plt.subplots(figsize=(10,6))

# Barras (importância)
ax1.bar(df_import["Feature"],         df_import["Importância"],
color="skyblue")
ax1.set_ylabel("Importância")
ax1.tick_params(axis="x", rotation=45)

# Linha acumulada (eixo secundário)
ax2 = ax1.twinx()
ax2.plot(df_import["Feature"],    df_import["%    Acumulado"],
color="red", marker="o", linestyle="-")
ax2.set_ylabel("% Acumulado")

# Linha de referência 80%
```

```
ax2.axhline(80, color="gray", linestyle="--")

# Título
plt.title("Gráfico de Pareto - Importância das Features
(XGBoost)")
plt.tight_layout()
plt.show()

"""# Otimizacao Banesian Search



"""

df = df_raw.copy()
df = pd.get_dummies(df, columns=['UF'], prefix='UF')
#df.drop(columns=['populacao'], inplace=True)
#df.drop(columns=['populacao', 'UF'], inplace=True)
#df['taxa_mortalidade_lag_1']                                    =
df['taxa_mortalidade'].shift(1)
#df.dropna(inplace=True)

X = df.drop("taxa_mortalidade", axis=1)

y = df["taxa_mortalidade"]

cond_treino = (X["ano"] <= 2023) | ((X["ano"] == 2024) &
(X["quad"] == 1))

# separa conjuntos
X_train = X[cond_treino]
y_train = y[cond_treino]

X_test = X[~cond_treino]
y_test = y[~cond_treino]

def add_quad_trig(df_in):
    out = df_in.copy()
    out["quad"] = out["quad"].astype(int)
    # quadrimestre = 3 ciclos/ano -> divisor = 3
    out["quad_sin"] = np.sin(2 * np.pi * out["quad"] / 3)
    out["quad_cos"] = np.cos(2 * np.pi * out["quad"] / 3)
    out = out.drop(columns=["quad"])
    return out

X_train = add_quad_trig(X_train)
X_test  = add_quad_trig(X_test)

# Definindo o modelo base
# Parâmetros fixos podem ser colocados aqui
xgb_reg = xgb.XGBRegressor(
```

```
    objective='reg:squarederror',
    eval_metric='rmse',
    n_jobs=-1,
    random_state=42
)

# Definindo o espaço de busca dos hiperparâmetros
#   Usamos   objetos   do   scikit-optimize   (Real,   Integer,
Categorical)
search_spaces = {
    'learning_rate': Real(1e-3, 0.3, 'log-uniform'),
    'n_estimators': Integer(100, 5000, 'log-uniform'),
    'max_depth': Integer(2, 30, 'uniform'),
    "min_child_weight": Integer(1.0, 50.0, 'uniform'),
    'subsample': Real(0.5, 1.0, 'uniform'),
    'colsample_bytree': Real(0.5, 1.0, 'uniform'),
    'reg_alpha': Real(0.0, 10.0, 'uniform'),
    'reg_lambda': Real(1e-3, 100.0, 'log-uniform'),
    'gamma': Real(0.0, 10.0, 'uniform'),
    "max_bin": Integer(32, 512, 'uniform'),
    #'booster': Categorical(['gbtree', 'dart']),
    'colsample_bylevel': Real(0.3, 1.0, 'uniform'),
}

# Usar apenas 2 splits para maximizar o tamanho do primeiro
fold de treino
tscv = TimeSeriesSplit(n_splits=2)

# Configurando a busca Bayesiana
bayes_search = BayesSearchCV(
    estimator=xgb_reg,
    search_spaces=search_spaces,
      n_iter=100,   # Número de combinações de parâmetros a
testar
    scoring='r2', # Métrica a ser otimizada
    cv=tscv,       # Número de folds para validação cruzada
    n_jobs=-1,  # Use todos os cores do processador
    verbose=1,
    random_state=42
)

print("Iniciando a busca Bayesiana...")
# Executando a busca no conjunto de treino
bayes_search.fit(X_train, y_train)

# Imprimindo os melhores hiperparâmetros encontrados
print("Melhores hiperparâmetros encontrados:")
print(bayes_search.best_params_)

model = bayes_search.best_estimator_
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)

# Criar modelo (regressão porque o target é numérico)
model = XGBRegressor(
    colsample_bylevel=0.3,
    colsample_bytree=1.0,
    gamma=2.3526454841602398,
    learning_rate=0.001,
    max_bin=512,
    max_depth=30,
    min_child_weight=29,
    n_estimators=4798,
    reg_alpha=8.512582403794102,
    reg_lambda=100.0,
    subsample=1.0
)

# Treinar
model.fit(X_train, y_train)

"""OrderedDict({'colsample_bylevel': 0.3, 'colsample_bytree':
1.0, 'gamma': 2.3526454841602398, 'learning_rate': 0.001,
'max_bin': 512, 'max_depth': 30, 'min_child_weight': 29,
'n_estimators': 4798, 'reg_alpha': 8.512582403794102,
'reg_lambda': 100.0, 'subsample': 1.0})


"""

y_pred = model.predict(X_test)

# Avaliação
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print("MSE:", mse)
print("RMSE:", rmse)
print("R²:", r2)
print("MAPE:", mape)

"""MSE: 4.984115123748779
RMSE: 2.232513185571091
R²: 0.03547024726867676
MAPE: 0.2620323598384857
"""

# Plot da importância das features
plt.figure(figsize=(8,5))
plt.barh(X_test.columns, model.feature_importances_)
```

```
plt.xlabel("Importância")
plt.ylabel("Variável")
plt.title("Importância das Features - XGBoost")
plt.show()

# Cria DataFrame com as features e suas importâncias
df_import = pd.DataFrame({
    "Feature": X_test.columns,
    "Importância": model.feature_importances_
})

# Ordena do maior para o menor
df_import       =       df_import.sort_values(by="Importância",
ascending=False)

# Calcula % acumulada
df_import["%         Acumulado"]         =         100         *
df_import["Importância"].cumsum()                                /
df_import["Importância"].sum()

# Plot
fig, ax1 = plt.subplots(figsize=(10,6))

# Barras (importância)
ax1.bar(df_import["Feature"],         df_import["Importância"],
color="skyblue")
ax1.set_ylabel("Importância")
ax1.tick_params(axis="x", rotation=45)

# Linha acumulada (eixo secundário)
ax2 = ax1.twinx()
ax2.plot(df_import["Feature"],    df_import["%    Acumulado"],
color="red", marker="o", linestyle="-")
ax2.set_ylabel("% Acumulado")

# Linha de referência 80%
ax2.axhline(80, color="gray", linestyle="--")

# Título
plt.title("Gráfico  de  Pareto  -  Importância  das  Features
(XGBoost)")
plt.tight_layout()
plt.show()

# garante que os dados estão alinhados por índice
y_test = pd.Series(y_test).reset_index(drop=True)
y_pred = pd.Series(y_pred).reset_index(drop=True)

plt.figure(figsize=(9, 5))
plt.plot(y_test, label="Real", marker="o", linestyle="-")
plt.plot(y_pred, label="Previsto", marker="x", linestyle="--")
```

```
plt.xlabel("Observações (tempo)")
plt.ylabel("Taxa de Mortalidade (original)")
plt.title("Regressão Linear Múltipla — Real vs Previsto
(desnormalizado)")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)
plt.tight_layout()
plt.show()

import numpy as np
import matplotlib.pyplot as plt

# --- 1) Previsões (normalizadas)
y_pred_norm = model.predict(X_test)


## Recupera o quadrimestre (você o removeu do X final, mas
podemos puxar do X_test_t)
# Caso X_test_t ainda exista:
quad_real = X_test_t.loc[X_test.index, "quad"]

# --- 4) Monta DataFrame com os valores desnormalizados
df_plot = pd.DataFrame({
    "ano_real": ano_real,
    "quad": quad_real,
    "Real": y_real,
    "Previsto": y_pred
})

# --- 5) Cria chave temporal combinando ano e quadrimestre
(ex: 2023.1, 2023.2, 2023.3)
df_plot["periodo"]                                          =
df_plot["ano_real"].astype(int).astype(str)     +     "."     +
df_plot["quad"].astype(str)

# --- 6) Calcula médias por período
df_media          =          df_plot.groupby("periodo")[["Real",
"Previsto"]].mean().reset_index()

# Ordena os períodos (para o eixo do gráfico)
df_media = df_media.sort_values("periodo")

# --- 7) Plot comparativo de linha
plt.figure(figsize=(9,5))
plt.plot(df_media["periodo"],  df_media["Real"],  label="Real",
marker="o")
plt.plot(df_media["periodo"],             df_media["Previsto"],
label="Previsto", marker="x")
plt.xlabel("Período (Ano.Quadrimestre)")
plt.ylabel("Taxa de Mortalidade (original)")
```

```
plt.title("Média por Quadrimestre - Real vs Previsto")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.4)
plt.tight_layout()
plt.show()

"""# XGboost Nao Otimizado


"""

df = df_raw.copy()
df.drop(columns=['populacao'], inplace=True)
df = pd.get_dummies(df, columns=['UF'], prefix='UF')

X = df.drop("taxa_mortalidade", axis=1)
y = df["taxa_mortalidade"]

# condição de corte
cond_treino = (X["ano"] <= 2023) | ((X["ano"] == 2024) &
(X["quad"] == 1))

# separa conjuntos
X_train = X[cond_treino]
y_train = y[cond_treino]

X_test = X[~cond_treino]
y_test = y[~cond_treino]

# Criar modelo (regressão porque o target é numérico)
model = XGBRegressor(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=4,
    random_state=42,
    enable_categorical = True
)

# Treinar
model.fit(X_train, y_train)

import math
# Previsões
y_pred = model.predict(X_test)

# Avaliação
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
```

```
print("MSE:", mse)
print("RMSE:", rmse)
print("R²:", r2)
print("MAPE:", mape)

"""# Sarimax(Arima Com Suporte a Multivariado)"""

df = df_raw.copy()

quad_to_month = {1: 1, 2: 5, 3: 9}
df["ds"] = pd.to_datetime({
    "year": df["ano"].astype(int),
    "month": df["quad"].map(quad_to_month).astype(int),
    "day": 1
})

# garante ordenação temporal e índice regular
df = df.sort_values("ds").set_index("ds")

y = df["taxa_mortalidade"].astype(float)

# 3) Split temporal (treino: até 2024-Q1; teste: depois)
cond_treino = (df["ano"] <= 2023) | ((df["ano"] == 2024) &
(df["quad"] == 1))

y_train_sarima = y[cond_treino]
y_test_sarima  = y[~cond_treino]

# Exemplo de ordens; você pode fazer uma busca de grade depois
order = (1,1,1)               # ARIMA(p,d,q)
seasonal_order   =   (1,1,1,3)             #   S(P,D,Q,   s=3
quadrimestres/ano)

model_sarima = SARIMAX(
    endog=y_train,
    order=order,
    seasonal_order=seasonal_order,
    enforce_stationarity=False,
    enforce_invertibility=False
)
res = model_sarima.fit(disp=False)

# Forecast para o tamanho do teste
y_pred_sarima = res.forecast(steps=len(y_test))
# Alinha índices
y_pred_sarima.index = y_test_sarima.index

# Métricas em escala original
mse = mean_squared_error(y_test_sarima, y_pred_sarima)
rmse = math.sqrt(mse)
r2 = r2_score(y_test_sarima, y_pred_sarima)
```

```
mape      =      mean_absolute_percentage_error(y_test_sarima,
y_pred_sarima)

print(f"MSE: {mse:.4f} | RMSE: {rmse:.4f} | R²: {r2:.4f} |
MAPE: {mape:.4%}")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# ========================
# 1) Normaliza y_pred_sarima
# ========================
sar = y_pred_sarima.copy()

# Se vier como Series, vira DataFrame
if isinstance(sar, pd.Series):
    sar = sar.to_frame(name="predicted_mean")

# Tenta detectar a coluna/índice de datas
dt = None
if "ds" in sar.columns:
    dt = pd.to_datetime(sar["ds"], errors="coerce")
else:
    # tenta índice
    dt_try = pd.to_datetime(sar.index, errors="coerce")
    if pd.isna(dt_try).all():
         # última tentativa: procurar uma coluna "date", "data"
etc.
        cand = None
        for c in sar.columns:
            lc = c.lower()
                 if lc in ("ds", "date", "data", "datetime",
"timestamp"):
                cand = c
                break
        if cand is None:
               raise ValueError("Não encontrei coluna/índice de
data em y_pred_sarima (esperei 'ds' ou índice datetime).")
        dt = pd.to_datetime(sar[cand], errors="coerce")
    else:
        dt = dt_try

sar = sar.assign(ds=dt).dropna(subset=["ds"])

# Garante coluna de valor: predicted_mean
if "predicted_mean" not in sar.columns:
    # se o nome for outro, pega a 1ª numérica
                                          num_cols           =
sar.select_dtypes(include="number").columns.tolist()
    if not num_cols:
```

```
            raise ValueError("y_pred_sarima não tem coluna
numérica com as previsões.")
    sar = sar.rename(columns={num_cols[0]: "predicted_mean"})

# =========================
# 2) Extrai ano/quad e agrega
# =========================
sar["ano"] = sar["ds"].dt.year.astype(int)
sar["quad"] = ((sar["ds"].dt.month - 1) // 4 + 1).astype(int)
# 1: jan-abr, 2: mai-ago, 3: set-dez

sarima_agg = (
                              sar.groupby(["ano",       "quad"],
as_index=False)["predicted_mean"]
       .mean()
       .sort_values(["ano", "quad"])
)
sarima_agg["periodo"] = sarima_agg["ano"].astype(str) + "-Q" +
sarima_agg["quad"].astype(str)

# =========================
# 3) Gráfico com histórico, ML e SARIMA
# =========================
plt.figure(figsize=(10, 5))

plt.plot(df_hist["periodo"],       df_hist["taxa_mortalidade"],
marker="o", label="Real (Histórico)")
plt.plot(df_media_prev["periodo"],     df_media_prev["y_pred"],
marker="o", label="Previsto (ML)")
plt.plot(sarima_agg["periodo"],   sarima_agg["predicted_mean"],
marker="o", label="Previsto (SARIMA)")

# Opcional: linha vertical marcando início do teste (se já
tiver df_media_prev)
if not df_media_prev.empty:
    inicio_teste = df_media_prev["periodo"].min()
        plt.axvline(inicio_teste,  linestyle="--",  alpha=0.7,
label=f"Início Predicoes ({inicio_teste})")

plt.title("Série Temporal — Mortalidade Real vs XGboost vs
SARIMAX", fontsize=14)
plt.xlabel("Período (Ano-Quadrimestre)", fontsize=12)
plt.ylabel("Taxa de Mortalidade Neonatal", fontsize=12)
plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.6)
plt.legend()
plt.tight_layout()
plt.show()

"""# Regressao Linear Multipla"""
```

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler    # faltava
importar
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_percentage_error
import math

# --- 1. Normaliza o y (target)
scaler_y = MinMaxScaler()  # ou StandardScaler()
y_train_scaled                                              =
scaler_y.fit_transform(y_train.values.reshape(-1, 1)).ravel()
y_test_scaled  =  scaler_y.transform(y_test.values.reshape(-1,
1)).ravel()

# --- 2. Cria e treina o modelo de regressão linear múltipla
lin_model = LinearRegression()
lin_model.fit(X_train, y_train_scaled)

# --- 3. Faz previsões (ainda normalizadas)
y_pred_norm = lin_model.predict(X_test)

# --- 4. Desnormaliza as previsões e valores reais
y_pred   =   scaler_y.inverse_transform(y_pred_norm.reshape(-1,
1)).ravel()
y_real  =  scaler_y.inverse_transform(y_test_scaled.reshape(-1,
1)).ravel()

# --- 5. Avalia as métricas
mse = mean_squared_error(y_real, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_real, y_pred)
mape = mean_absolute_percentage_error(y_real, y_pred)

print("===== Regressão Linear Múltipla =====")
print(f"MSE:   {mse:.4f}")
print(f"RMSE:  {rmse:.4f}")
print(f"R²:    {r2:.4f}")
print(f"MAPE:  {mape:.4%}")
```

APÊNDICE D - ARTIGO

Desenvolvimento de um ETL para Dados de Mortalidade NeoNatal e um Modelo Multivariado de Aprendizado de Máquina para Predição da Taxa de Mortalidade Neonatal

Pedro M. Pereira[1]

[1] Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

pedro.matiucci@grad.ufsc.br

Abstract. *Neonatal mortality is a central indicator for assessing the effectiveness of public health policies and the performance of Primary Health Care, as well as being part of the United Nations' 2030 Agenda targets. However, official data are typically released with a delay of about one year, which hinders timely analyses. This study developed an ETL pipeline to integrate data from SIM, SINASC, and SISAB, and built a multivariate predictive model based on XGBoost to estimate neonatal mortality rates in large municipalities on a quadrimester basis. Using the CRISP-DM methodology and Bayesian optimization, the model was compared to SARIMA and demonstrated superior performance across all evaluated metrics.*

Resumo. *A taxa de mortalidade neonatal é um indicador central para avaliar a efetividade das políticas públicas e o desempenho da Atenção Primária à Saúde, além de integrar as metas da Agenda 2030 da ONU. Contudo, os dados oficiais apresentam defasagem de cerca de um ano, dificultando análises oportunas. Este trabalho desenvolveu um pipeline de ETL para integrar dados do SIM, SINASC e SISAB e construiu um modelo preditivo multivariado baseado em XGBoost para estimar a taxa de mortalidade neonatal em cidades de grande porte, com periodicidade quadrimestral. Utilizando a metodologia CRISP-DM e otimização bayesiana, o modelo foi comparado ao SARIMA e apresentou desempenho superior em todas as métricas avaliadas.*

1. Introdução

A taxa de mortalidade neonatal é um indicador central para o monitoramento da qualidade da atenção à saúde materno-infantil, orientando decisões de gestão e a alocação de recursos públicos. Esse desfecho é utilizado para acompanhar a evolução e a efetividade das políticas de saúde, sendo um dos critérios adotados pelo Programa Previne Brasil para a distribuição de receitas entre municípios (Ministério da Saúde, 2024). Sua relevância também se insere no cenário internacional, uma vez que a Agenda 2030 da Organização das Nações Unidas estabelece como meta reduzir a taxa para menos de 12 mortes a cada mil nascidos vivos, reforçando a importância global do tema (Garcia et al., 2023).

Embora avanços tenham sido alcançados, o Brasil ainda apresenta desigualdades regionais significativas e enfrenta desafios referentes à qualidade e à disponibilidade dos dados oficiais. As informações de mortalidade e natalidade provenientes de sistemas como o SINASC (Sistema de Informações sobre Nascidos Vivos) e o SIM (Sistema de Informações sobre Mortalidade) podem levar até dois anos para serem consolidadas e

disponibilizadas, criando uma defasagem que limita o monitoramento em tempo oportuno e dificulta respostas rápidas no âmbito da saúde pública.

Diante dessas limitações, torna-se necessário explorar abordagens capazes de antecipar tendências e reduzir a dependência da consolidação tardia dos dados oficiais. No contexto deste estudo, técnicas de aprendizado de máquina são aplicadas à previsão de séries temporais da taxa de mortalidade neonatal, oferecendo potencial para capturar padrões complexos e produzir estimativas mais robustas que métodos estatísticos tradicionais. Isso favorece análises preditivas mais rápidas e precisas, ampliando a capacidade de intervenção de gestores e pesquisadores.

Nesse sentido, este trabalho propõe o desenvolvimento de um processo automatizado de ETL para tratamento e preparação dos dados, bem como a construção de um modelo preditivo multivariado para estimar a taxa de mortalidade neonatal nos municípios brasileiros. Do ponto de vista tecnológico, a integração entre engenharia de dados e modelagem preditiva resulta em um pipeline reprodutível e escalável, demonstrando a aplicabilidade da ciência de dados no campo da saúde pública. Do ponto de vista social, os resultados esperados têm o potencial de apoiar estratégias locais e regionais de prevenção e acompanhamento neonatal, promovendo decisões mais rápidas e assertivas no âmbito do Sistema Único de Saúde.

2. Trabalhos Relacionados

A predição de desfechos relacionados à mortalidade neonatal e infantil por meio de técnicas de aprendizado de máquina tem recebido atenção crescente na literatura, embora ainda apresente limitações quanto ao escopo metodológico, diversidade de dados e capacidade de generalização. Nesta seção, são discutidos três estudos diretamente relacionados ao tema deste trabalho: um estudo aplicado com dados de São Paulo(Batista et al., 2021), uma revisão sistemática recente (Mangold et al., 2021) e uma tese de doutorado com foco regional (Aguiar, 2019).

1. Batista et al. (2021): representa uma das iniciativas mais robustas no contexto brasileiro ao aplicar modelos de aprendizado de máquina para predição de mortalidade neonatal. Utilizando dados do SINASC e do SIM do município de São Paulo, os autores compararam diferentes algoritmos e observaram desempenho superior do XGBoost, que apresentou AUC de 0,971. O estudo destaca a relevância de variáveis como peso ao nascer, idade gestacional e Apgar, mas suas conclusões permanecem restritas a um único contexto urbano, sem validação externa ou avaliação em diferentes realidades regionais.

2.Mangold et al. (2021): reúne evidências internacionais sobre o uso de modelos de aprendizado de máquina na predição de mortalidade neonatal. Os autores identificam grande heterogeneidade metodológica entre os estudos, destacando a falta de padronização, a escassez de validação externa e a ausência de avaliação da calibração dos modelos. A revisão reforça o potencial das técnicas de ML, mas aponta limitações que dificultam sua adoção prática em sistemas de saúde.

3.Aguiar (2019): representa uma das poucas iniciativas nacionais voltadas à mortalidade infantil com enfoque regional. O estudo aplicou redes neurais para prever óbitos infantis no estado do Ceará, alcançando elevado desempenho. Embora relevante, a pesquisa é

limitada pela escala territorial e pela falta de integração com bases nacionais mais amplas, como SISAB, SIM e SINASC em nível nacional, o que reduz sua aplicabilidade em contextos mais diversos.

Com base na análise da literatura existente é possível observar que concentra-se majoritariamente em modelos de classificação individual (óbito/sobrevivência), enquanto o presente estudo adota uma abordagem de regressão temporal agregada por município, mais adequada ao monitoramento e à vigilância em saúde pública. Além disso, observa-se ausência de validação temporal rigorosa, poucos estudos com múltiplas fontes de dados públicas e integrações com indicadores assistenciais da Atenção Primária, como aqueles fornecidos pelo SISAB.

Outro ponto é que nenhum dos estudos analisados aborda o problema da defasagem informacional dos dados de mortalidade no Brasil, que frequentemente leva mais de um ano até sua consolidação. Essa lacuna metodológica abre espaço para abordagens baseadas em nowcasting e forecasting, capazes de mitigar atrasos e apoiar decisões mais oportunas.

O presente trabalho avança a literatura ao:

• propor predição multivariada da taxa municipal de mortalidade neonatal em séries temporais quadrimestrais;

• integrar indicadores assistenciais do SISAB como variáveis exógenas;

• aplicar validação temporal com comparação entre modelos estatísticos (SARIMA) e de aprendizado de máquina (XGBoost);

• desenvolver um pipeline de ETL automatizado, escalável e reprodutível;

• abordar a defasagem informacional por meio de uma arquitetura voltada a nowcasting.

3. Metodologia

A metodologia adotada neste estudo foi organizada em duas frentes complementares: o desenvolvimento do processo de ETL (Extract, Transform and Load) e a construção do modelo de aprendizado de máquina para predição da taxa de mortalidade neonatal.

3.1 Processo de ETL

O processo de ETL foi desenvolvido em Python, com scripts responsáveis pela extração automática dos dados de natalidade e mortalidade provenientes dos sistemas SINASC e SIM. As informações foram obtidas por meio de requisições HTTP a URLs públicas, seguindo abordagem semelhante à descrita por Cesar (2024), cujo trabalho serviu como referência inicial. Para isso, foi realizado um fork do repositório público disponibilizado pelo autor e, a partir dele, foram implementadas adaptações necessárias ao escopo deste estudo.

O link utilizado como base é:

https://codigos.ufsc.br/marco.cesar/yll_time_series_machine_learning

Já para os dados do SISAB, que não possui APIs ou repositórios padronizados para download. Foi desenvolvido um web scraper específico para extrair informações diretamente do portal público do SISAB, respeitando suas limitações estruturais e de acesso.

Após a extração, todas as bases foram tratadas, padronizadas e integradas. Os dados resultantes foram armazenados no Google BigQuery, o que facilitou o processamento, o versionamento e o treinamento dos modelos. O código referente a essa etapa está disponível em:

https://codigos.ufsc.br/tcc_etl_modelo_multivariado_previsao_mortalidade/etl_tcc/-/tree/763a67c1321ddaf6c8fbd27d044356c3c6604185/

3.2 Desenvolvimento do Modelo Preditivo

Uma segmentação da metodologia CRISP-DM foi utilizada como referência para estruturar as etapas do projeto, contemplando 6 etapas: compreensão do negócio, compreensão dos dados, preparação dos dados, modelagem e avaliação.

3.2.1 Compreensão do negócio

Nesta etapa foram identificadas as fontes de dados necessárias, as variáveis disponíveis nos sistemas públicos e a frequência temporal de cada base. Observou-se que os indicadores do SISAB são disponibilizados em periodicidade quadrimestral no período de 2022 a 2025. Para garantir compatibilidade entre as fontes, os dados de mortalidade e natalidade foram convertidos para a mesma granularidade quadrimestral adotada pelo SISAB.

3.2.2 Compreensão dos dados

A análise inicial teve como objetivo avaliar a estrutura, consistência e disponibilidade das variáveis. Foram consideradas três bases principais: SIM, SINASC e SISAB. Os indicadores provenientes do SISAB selecionados devido à sua associação com desfechos neonatais foram:

- proporção de gestantes com pelo menos seis consultas de pré-natal;
- proporção de gestantes com exames de sífilis e HIV;
- proporção de gestantes com atendimento odontológico;
- proporção de mulheres com coleta de exame citopatológico.

Foi conduzida uma análise exploratória para identificar padrões temporais, sazonalidade, relação entre óbitos e nascidos vivos e eventuais inconsistências entre as fontes.

3.2.3 Preparação dos dados

A etapa de preparação foi realizada em Python, utilizando as tabelas já pré-processadas no ETL. Primeiramente, os dados foram agregados para a frequência quadrimestral. Em seguida, foi aplicado um filtro para incluir apenas municípios de grande porte, definidos como aqueles com população entre 100 mil e 900 mil habitantes.

Essa decisão metodológica foi tomada porque municípios menores apresentaram alta proporção de dados ausentes e inconsistências, como períodos com número de óbitos

superior ao número de nascidos vivos, o que inviabilizaria o treinamento do modelo. Ressalta-se que o ETL completo continua processando todos os municípios e o filtro é aplicado apenas na etapa de modelagem.

Os seguintes procedimentos adicionais foram realizados:

• remoção de valores atípicos pelo método do intervalo interquartil (IQR);

• exclusão de registros inconsistentes;

• remoção de municípios com mais de 80% de ausência na variável-alvo;

• recálculo da taxa de mortalidade neonatal por município e quadrimestre;

• transformação da variável "quadrimestre" em representação trigonométrica (seno e cosseno) para capturar sazonalidade;

• inclusão de variáveis preditoras correspondentes aos indicadores assistenciais do SISAB.

Após essas etapas, o conjunto final de dados totalizou 1.751 registros e 9 variáveis no período entre 2022 e o primeiro quadrimestre de 2025.

3.2.4 Modelagem

Foram testados três modelos preditivos: Regressão Linear Múltipla, SARIMAX e XGBoost. Todos foram ajustados sobre a mesma base de dados e avaliados pelas mesmas métricas, garantindo comparação direta entre os resultados.

As métricas utilizadas foram MSE, RMSE, R^2 e MAPE. O XGBoost foi selecionado para aprofundamento devido ao seu desempenho superior e à capacidade de modelar relações não lineares.

A otimização dos hiperparâmetros foi realizada por meio de Otimização Bayesiana utilizando BayesSearchCV. Essa abordagem probabilística reduz o custo computacional e apresenta maior eficiência de busca quando comparada aos métodos tradicionais Grid Search e Random Search.(TURNER et al., 2021).

3.2.5 Avaliação

A avaliação final foi realizada utilizando como conjunto de teste. Os dados anteriores foram utilizados exclusivamente para treinamento, respeitando à ordem temporal e evitando vazamento de informação.

As métricas MSE, RMSE, R^2 e MAPE foram utilizadas de forma complementar, sendo o MAPE definido como métrica principal por representar o erro relativo em termos percentuais. Para evitar sobreajuste e garantir robustez, foi empregada validação cruzada temporal com dois splits utilizando TimeSeriesSplit.

O modelo selecionado foi treinado integralmente no conjunto de treino e posteriormente avaliado no conjunto de teste. Os resultados quantitativos referentes a cada métrica são apresentados na seção de resultados.

4. Resultados e Discussões

O processo de ETL desenvolvido apresentou desempenho satisfatório na extração, padronização e integração dos dados provenientes dos sistemas SIM, SINASC e SISAB. A principal dificuldade esteve relacionada ao SISAB, que não disponibiliza APIs nem arquivos tabulados padronizados, o que exigiu a construção de um web scraper específico para a coleta automatizada das informações. O pipeline também permite ajustar dinamicamente o período de coleta e realizar atualizações periódicas sem necessidade de modificações estruturais.

Uma decisão metodológica relevante foi armazenar os dados em formato totalmente desagregado, com um registro por linha e sem agregações prévias. Essa abordagem amplia a flexibilidade analítica e possibilita que diferentes estudos utilizem o mesmo conjunto de dados segundo suas próprias necessidades, sem restrições impostas pela arquitetura do ETL.

Na etapa de análise exploratória, buscou-se avaliar a qualidade dos dados, identificar padrões temporais e compreender a relação entre os indicadores assistenciais e a mortalidade neonatal. Observou-se que a taxa de mortalidade apresenta heterogeneidade expressiva, variando de 3,75 a 13,94 óbitos por mil nascidos vivos. Os indicadores do SISAB também apresentaram alta variabilidade, sugerindo diferenças significativas na cobertura assistencial entre municípios.

A análise temporal indicou a presença de um padrão sazonal consistente, com valores mais baixos no segundo quadrimestre e elevação progressiva até o terceiro quadrimestre. Esse comportamento confirma a necessidade de modelos capazes de capturar componentes sazonais além da tendência. A decomposição da série reforçou essa evidência ao revelar tendência, sazonalidade definida e resíduo estruturado, caracterizando um processo não estacionário e dependente de tempo.

Três modelos iniciais foram avaliados: Regressão Linear Múltipla, SARIMA e XGBoost. Entre eles, o XGBoost apresentou o melhor desempenho geral ainda antes da otimização de hiperparâmetros. A partir desse resultado, aplicou-se a Otimização Bayesiana para aprimorar o modelo, que posteriormente foi comparado ao SARIMA como baseline. Na Tabela 1 as métricas finais demonstraram desempenho superior do XGBoost em todas as métricas avaliadas.

Tabela 1 – Comparativo de desempenho entre SARIMA e XGBoost: MSE, RMSE, R^2 e MAPE

Métrica	XGboost	Sarima
MSE (Mean Squared Error)	5,51	5.79
RMSE (Root Mean Squared Error)	2,35	2.4

R^2 (Coeficiente de determinação)	0,03	-0.02
MAPE (Mean Absolute Percentage Error)	0,28	0.30

Após a avaliação global, procedeu-se à análise regional. Observou-se que a tendência de aproximação dos valores extremos à média ocorreu especialmente nas regiões Nordeste e Centro-Oeste, que concentram municípios com taxas mais distantes do padrão nacional. Esse efeito decorre da baixa representatividade de valores extremos no conjunto de treinamento. A análise das métricas regionais evidenciou desempenho superior nas regiões Sul, Sudeste e Norte. A Tabela 2 demonstra os resultados obtidos na análise por região.

Tabela 2 – Métricas de erro por região e conjuntos de estados

Região	MSE	RMSE	R^2	MAPE
Sul	3.63	1.90	0.006	0.26
Sudeste	5.38	2.32	0.020	0.27
Centro-Oeste	6.74	2.59	-0.099	0.28
Nordeste	7.22	2.68	-0.048	0.28
Norte	4.44	2.10	0.0617	0.26

Apesar dessas limitações, o modelo mostrou capacidade consistente de capturar tendência e sazonalidade, o que o torna útil para fins de nowcasting da mortalidade neonatal. A principal limitação está relacionada ao curto período temporal disponível no SISAB, restrito a 2022–2025. A ampliação futura dessa série permitirá reavaliar e possivelmente aprimorar o desempenho do modelo, aumentando sua capacidade de generalização.

5. Conclusão

O trabalho desenvolveu um pipeline de ETL e um modelo preditivo multivariado para estimar a taxa de mortalidade neonatal em municípios brasileiros a partir dos dados dos sistemas SIM, SINASC e SISAB. O ETL demonstrou robustez e flexibilidade, possibilitando a integração automatizada das bases e a atualização contínua dos períodos analisados. O modelo preditivo apresentou desempenho satisfatório, capturando tendências e padrões sazonais da mortalidade neonatal com maior precisão nas faixas mais comuns entre municípios de grande porte. Quando comparado ao modelo

tradicional SARIMA, o XGBoost mostrou desempenho superior em todas as métricas avaliadas. Os resultados reforçam a viabilidade da abordagem proposta e evidenciam o potencial do aprendizado de máquina para apoiar a tomada de decisão em saúde pública.

Referências

BRASIL. Ministério da Saúde. Previne Brasil: novo modelo de financiamento da Atenção Primária à Saúde. Disponível em: https://www.gov.br/saude/pt-br/composicao/saps/previne-brasil. Acesso em: 29 jun. 2025.

CESAR, Marco; SILVA, D. A. Modelo preditivo dos anos de vida perdidos por morte prematura para os municípios brasileiros de médio porte utilizando aprendizagem de máquina. Trabalho de Conclusão de Curso. Universidade Federal de Santa Catarina, [s.d.]. Disponível em: https://codigos.ufsc.br/marco.cesar/yll_time_series_machine_learning. Acesso em: 29 jun. 2025.

AGUIAR, Wellington Sousa. Desenvolvimento de modelos preditivos de mortalidade infantil com base em inteligência artificial no estado do Ceará. 2019. 163 f. Tese (Doutorado em Saúde Pública) - Faculdade de Medicina. Programa de Pós-Graduação em Saúde Pública, Universidade Federal do Ceará, Fortaleza, 2019. Disponível em: http://www.repositorio.ufc.br/handle/riufc/72867. Acesso em: 29 jun. 2025.

GARCIA, L. P.; SCHNEIDER, I. J. C.; DE OLIVEIRA, C.; et al. What is the impact of national public expenditure and its allocation on neonatal and child mortality? A machine learning analysis. BMC Public Health, v. 23, p. 793, 2023. Disponível em: https://doi.org/10.1186/s12889-023-15683-y. Acesso em: 29 jun. 2025.

MANGOLD, C.; ZORETIC, S.; THALLAPUREDDY, K.; MOREIRA, A.; CHORATH, K.; MOREIRA, A. Machine learning models for predicting neonatal mortality: A systematic review. Neonatology, v. 118, n. 4, p. 394–405, 2021. Disponível em: https://doi.org/10.1159/000516891. Acesso em: 29 jun. 2025.

Batista, A. F. M., Diniz, C. S. G., Bonilha, E. A., Kawachi, I., & Chiavegatto Filho, A. D. P. (2021). Neonatal mortality prediction with routinely collected data: a machine learning approach. BMC Pediatrics, 21(1). https://doi.org/10.1186/s12887-021-02788-9

Turner, R., Eriksson, D., Mccourt, M., Kiili, J., Xu, V. Z., Escalante, H. J., & Hofmann, K. (n.d.). Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. http://clopinet.com/isabelle/Projects/NIPS2006

MANGOLD, C.; ZORETIC, S.; THALLAPUREDDY, K.; MOREIRA, A.; CHORATH, K.; MOREIRA, A. Machine learning models for predicting neonatal mortality: A systematic review. Neonatology, v. 118, n. 4, p. 394–405, 2021. Disponível em: https://doi.org/10.1159/000516891. Acesso em: 5 jul. 2025.

MINISTÉRIO DA SAÚDE. Previne Brasil: novo modelo de financiamento da Atenção Primária à Saúde. Brasília: Ministério da Saúde, 2019.

UNITED NATIONS. Transforming our world: the 2030 Agenda for Sustainable Development. New York: United Nations, 2015.