Universidade Católica do Salvador
Bacharelado em Engenharia de Software

Beatriz Nascimento Gibaut Santos
Gabriel de Abreu Farias Azevedo
Lucas Braga Carneiro
Rebeca Bezerra Gonçalves dos Santos
Sarah Evellyn Ferreira Nogueira

Previsão de Ocupação em Hospitais Públicos Utilizando Modelos de Séries Temporais e Aprendizado de Máquina

Salvador
2025

Beatriz Nascimento Gibaut Santos
Gabriel de Abreu Farias Azevedo
Lucas Braga Carneiro
Rebeca Bezerra Gonçalves dos Santos
Sarah Evellyn Ferreira Nogueira

Previsão de Ocupação em Hospitais Públicos Utilizando Modelos de Séries Temporais e Aprendizado de Máquina

Trabalho de Conclusão de Curso apresentado à Universidade Católica do Salvador como parte dos requisitos necessários para a obtenção do Título de Engenheiro de Software.
Orientador: Prof. Dr. Marco Simões

Universidade Católica do Salvador

Salvador
2025

Beatriz Nascimento Gibaut Santos
Gabriel de Abreu Farias Azevedo
Lucas Braga Carneiro
Rebeca Bezerra Gonçalves dos Santos
Sarah Evellyn Ferreira Nogueira

Previsão de Ocupação em Hospitais Públicos Utilizando Modelos de Séries Temporais e Aprendizado de Máquina

Trabalho de Conclusão de Curso apresentado à Universidade Católica do Salvador como requisito parcial para a obtenção do título de Engenheiro de Software.

Salvador, 15 de janeiro de 2026
Banca Examinadora:

Prof. Dr. Marco Simões
Universidade Católica do Salvador
Orientador

Prof. Me. Segundo professor
Universidade Católica do Salvador

Prof. Me. Terceiro professor
Universidade Católica do Salvador

Resumo

A previsão da demanda por internações hospitalares é fundamental para o planejamento de leitos, insumos e equipes, sobretudo em contextos marcados por sazonalidade e surtos epidemiológicos. Este trabalho teve como objetivo avaliar e comparar diferentes modelos de previsão aplicados a série temporal de internações mensais no Brasil, construída a partir de dados do SIHSUS integrados a informações epidemiológicas do SINAN. Inicialmente, foram realizadas análises estatísticas descritivas para caracterizar o comportamento da série e identificar padrões de tendência, sazonalidade e possíveis valores extremos. Em seguida, procedeu-se a análise inferencial e a construção do conjunto de dados experimental, incluindo tratamento de valores ausentes, geração de defasagens e análise de sensibilidade ao tratamento de outliers. No experimento foram ajustados modelos estatísticos clássicos e modelos de aprendizado de máquina, avaliados por meio de RMSE, MAE e MAPE em um esquema de validação temporal com divisão de dados para treinamento e para teste. Os resultados indicaram que a Regressão Linear com três defasagens apresentou o melhor desempenho preditivo, superando tanto o modelo SARIMA quanto o Random Forest, especialmente em termos de erro percentual médio. A análise de sensibilidade mostrou ainda que o desempenho do SARIMA é fortemente influenciado por valores extremos associados a surtos.

Palavras-Chave: 1. Previsão. 2. IA. 3. Séries temporais. 4. Aprendizado de máquina. 5. Internações hospitalares.

Abstract

Forecasting hospital admission demand is essential for planning beds, supplies, and staffing, especially in contexts marked by seasonality and epidemiological outbreaks. This study aimed to evaluate and compare different forecasting models applied to a monthly time series of hospital admissions in Brazil, constructed from SIHSUS data integrated with epidemiological information from SINAN. Initially, descriptive statistical analyses were performed to characterize the behavior of the series and identify patterns of trend, seasonality, and potential extreme values. Subsequently, inferential analysis and the construction of the experimental dataset were carried out, including the treatment of missing values, generation of lagged variables, and sensitivity analysis regarding outlier handling. In the experiment, classical statistical models and machine learning models were fitted and evaluated using RMSE, MAE, and MAPE within a temporal validation scheme with separate training and testing sets. The results indicated that Linear Regression with three lags achieved the best predictive performance, outperforming both the SARIMA model and Random Forest, especially in terms of mean percentage error. The sensitivity analysis also showed that the performance of SARIMA is strongly influenced by extreme values associated with outbreaks.

Keywords: 1. Forecasting. 2. AI. 3. Time series. 4. Machine learning. 5. Hospital admissions.

Lista de figuras

Lista de tabelas

Lista de Siglas e Abreviaturas

API	*Application Programming Interface*
ANN	Redes neurais artificiais
ARIMA	Média móvel integrada autoregressiva
BES	Bacharelado em Engenharia de Software
CNN	Convolutional Neural Networks
FTS	Series temporais fuzzy
IA	Inteligência artificial
LSTM	Long Short-Term Memory
MA	Média Móvel
MAD	Desvio absoluto mediano
MAE	Erro Médio Absoluto
MAPE	Erro percentual absoluto médio
ML	Aprendizado de máquina
RMSE	Raiz do Erro Quadrático Médio
SIHSUS	Sistema de Informações Hospitalares do SUS
SINAN	Sistema de Informação de Agravos de Notificação
SUS	Sistema único de saúde

Sumário

1 Introdução

A capacidade de gerir de forma eficiente os recursos humanos, materiais e financeiros de um sistema de saúde é essencial para garantir acesso oportuno e cuidado adequado à população. A utilização adequada de leitos, a organização de escalas de profissionais, o funcionamento das redes de urgência e a resposta a surtos epidemiológicos dependem de estimativas minimamente confiáveis da quantidade de internações esperadas ao longo do tempo. Em países com grande desigualdade social e heterogeneidade regional, como o Brasil, esses desafios se agravam, pois o volume de atendimento oscila fortemente entre municípios de diferentes portes, perfis assistenciais e contextos socioepidemiológicos.

No contexto brasileiro, o SIHSUS registra anualmente milhões de internações financiadas pelo SUS, enquanto o SINAN consolida notificações de agravos como dengue, influenza, meningite, leptospirose e doenças transmitidas por alimentos. Esses eventos epidemiológicos, quando evoluem para surtos, tendem a provocar aumentos abruptos na procura por serviços de saúde, afetando diretamente a taxa de ocupação hospitalar, a permanência média e a disponibilidade de leitos, com risco de colapso localizado da rede assistencial.

Neste trabalho, o foco empírico recai sobre a previsão de internações hospitalares mensais agregadas no Brasil, no período de 2020 a 2024, a partir da integração de dados do SIHSUS com agravos epidemiológicos do SINAN, recorte particularmente crítico por abranger a fase aguda e os desdobramentos da pandemia de COVID-19. Do ponto de vista da Engenharia de Software, lidar com essas bases públicas de saúde representa um desafio de engenharia de dados mais complexo do que simplesmente ajustar modelos estatísticos sobre um dataset pronto. As informações do SIHSUS e do SINAN são volumosas, heterogêneas e distribuídas em arquivos distintos, exigindo etapas explícitas de extração, limpeza, padronização de dados, tratamento de valores ausentes, agregação temporal e integração entre múltiplas chaves como unidade hospitalar, município, UF, ano, mês. Construir um pipeline robusto de dados, capaz de consolidar essas fontes em uma série temporal nacional de internações mensais associada a agravos epidemiológicos, constitui em si um problema relevante de Engenharia de Software aplicada à saúde, envolvendo decisões de projeto, organização modular de scripts e uso de técnicas de processamento em blocos (chunks) para contornar limitações de memória.

Tradicionalmente, previsões de demanda hospitalar foram realizadas com base em modelos estatísticos clássicos, como médias móveis, modelos autorregressivos e modelos ARIMA/SARIMA. Embora esses métodos sejam bem estabelecidos e interpre-

táveis, seu desempenho pode se deteriorar quando a série apresenta forte não linearidade, mudanças bruscas de comportamento, efeitos de surtos epidêmicos e variabilidade estrutural, como ocorre em séries curtas impactadas pela COVID-19. Com o avanço recente da Inteligência Artificial, modelos de aprendizado de máquina supervisionado, como Random Forest, passaram a ser explorados em previsão em saúde por sua maior capacidade de capturar relações não lineares e interações entre múltiplas variáveis explicativas.

Nesse cenário, surge a questão sobre quais tipos de modelos são mais adequados para prever internações hospitalares em uma série temporal curta, com ruptura estrutural pandêmica e forte presença de valores extremos. Este trabalho foca na comparação entre modelos estatísticos clássicos como SARIMA, médias móveis e regressão linear com defasagens e métodos de aprendizado de máquina supervisionado em particular, aplicados à série de internações mensais agregadas do Brasil, integrada a indicadores epidemiológicos derivados do SINAN. O desempenho dos modelos é avaliado por meio de métricas de erro como RMSE, MAE e MAPE, em um esquema de validação temporal que separa janelas de treinamento e de teste, tomando como referência modelos baseline ingênuos como Naive, Drift e médias móveis.

Embora a literatura sobre previsão de demanda hospitalar tenha avançado, ainda há lacunas importantes quando se considera o contexto do SUS. Diversos estudos utilizam bases prontas, de hospitais isolados ou registros de países de alta renda, sem enfrentar a complexidade de integrar, em escala nacional, dados administrativos e epidemiológicos provenientes de sistemas públicos como SIHSUS e SINAN. Além disso, são relativamente escassos os trabalhos que comparam, de forma sistemática, modelos como regressões lineares com defasagens e modelos mais complexos como Random Forest, em séries temporais curtas, fortemente influenciadas pela pandemia de COVID-19 e por surtos epidemiológicos. A lacuna não se restringe à escolha do algoritmo, mas inclui a ausência de descrições detalhadas de pipelines de extração, carregamento e transformação de dados a partir de fontes governamentais reais, diante desse cenário este trabalho formula a seguinte pergunta de pesquisa, entre modelos estatísticos clássicos e modelos de aprendizado de máquina supervisionado, qual modelo apresenta melhor desempenho preditivo para a série de internações hospitalares mensais agregadas do Brasil, no período de 2020 a 2024, construída a partir da integração de dados do SIHSUS e de agravos epidemiológicos do SINAN?

A hipótese de trabalho considerada é que modelos de aprendizado de máquina tendem a superar modelos estatísticos clássicos na tarefa de previsão, especialmente em séries complexas com efeitos de surtos e variáveis explicativas epidemiológicas. O sucesso dos modelos é avaliado pela capacidade de reduzir as métricas de erro (RMSE, MAE e MAPE) em relação a modelos de referência simples, como médias móveis e modelos Naive, em um esquema de validação temporal coerente com a na-

tureza sequencial dos dados.

Assim, a solução do problema de planejar recursos assistenciais em um cenário de alta variabilidade e incerteza, e considerando a ausência de estudos que integrem de forma explícita SIHSUS e SINAN para construir uma série nacional de internações mensais e comparar sistematicamente modelos simples e complexos em uma série curta com forte influência da COVID-19, este trabalho tem como objetivo geral desenvolver e avaliar um pipeline de engenharia de dados e de modelagem preditiva que integre dados do SIHSUS, CNES e SINAN, realize análises descritivas e inferenciais, e compare o desempenho de modelos estatísticos clássicos e de aprendizado de máquina na previsão de internações hospitalares mensais agregadas no Brasil. Ao explicitar o processo de integração de dados, as escolhas de modelagem e os critérios de avaliação, a pesquisa busca contribuir tanto para a literatura de previsão em saúde quanto para a prática de Engenharia de Software aplicada a sistemas de informação em saúde pública.

1.1 Aplicabilidade e Motivação

A área da saúde é um setor que ainda pode explorar bastante o horizonte de tecnologias aplicadas, desta forma uma delas é justamente utilizando Machine Learning para tomadas de decisões administrativas. O grande ponto de motivação deste trabalho foi utilizar tecnologia para impactar diretamente a sociedade, trazendo o poder de melhorar o gerenciamento de hospitais no quesito alocação de pessoas, deste modo distribuindo melhor os profissionais da saúde de acordo com a previsibilidade de demanda e atendendo melhor a população.

1.2 Objetivos

Avaliar e comparar o desempenho de diferentes métodos de previsão estatísticos, de machine learning e de deep learning, na estimativa mensal de internações hospitalares no Brasil, identificando o modelo mais eficiente e adequado para apoiar o planejamento e a gestão dos serviços de saúde.

1.3 Objetivos Específicos

- Integrar e consolidar dados do SIHSUS e do SINAN em uma base mensal contendo informações de internações hospitalares, variáveis estruturais e agravos epidemiológicos.

- Realizar a análise exploratória da base, avaliando completude, estrutura temporal, outliers, sazonalidade e padrões epidemiológicos relevantes.
- Aplicar testes de hipótese para verificar se surtos de doenças (como dengue e influenza) influenciam significativamente o volume de internações mensais.
- Desenvolver e treinar modelos estatísticos clássicos (média móvel, Holt-Winters e SARIMA) para previsão de demanda hospitalar.
- Desenvolver e treinar modelos de deep learning (LSTM e CNN 1D) para captura de padrões complexos e dependências temporais na série histórica.
- Identificar o modelo mais preciso para previsão de internações hospitalares.

2 Fundamentação Teórica

Ultimamente a preocupação com a ocupação hospitalar tem crescido de forma significativa, especialmente em contextos de superlotação em emergências e escassez de leitos nos sistemas públicos de saúde. Esse cenário tem levado diversos pesquisadores a buscar soluções baseadas em modelos preditivos, capazes de antecipar variações na demanda e assim auxiliando nas tomadas de decisões operacionais nos sistemas de saúde.

Diversos estudos, realizados em diferentes países, exploram desde métodos estatísticos de séries temporais até técnicas mais recentes de aprendizado de máquina, com o objetivo de prever o comportamento da ocupação hospitalar e otimizar fluxos assistenciais. De modo geral, os trabalhos revisados podem ser agrupados em três eixos, que serão explorados para fundamentar as escolhas metodológicas e a relevância do presente estudo: Modelos voltados à predição de fluxos e ocupação hospitalar; Pesquisas que analisam o impacto da superlotação e de políticas de tempo de permanência; Estudos que investigam o uso da inteligência artificial (IA) na gestão hospitalar.

2.1 Predição de Fluxos e Ocupação Hospitalar

Um dos trabalhos revisados que se destacou foi o de Jilani et al. (2019), que desenvolveu um modelo de previsão de atendimentos em departamentos de emergência utilizando FTS. Os autores compararam o desempenho desse modelo com os métodos tradicionais ARIMA e com redes neurais artificiais (ANN) em quatro hospitais públicos do Serviço nacional de saúde do no Reino Unido. O modelo fuzzy apresentou resultados superiores, com MAPE entre 2% e 4%, demonstrando maior precisão para horizontes de previsão de quatro semanas e quatro meses. Desta forma, os autores do trabalho concluíram a importância do projeto para o planejamento e gestão dos leitos hospitalares de forma mais eficiente.

Na mesma linha, Jiang et al. (2017) propuseram um modelo baseado em Deep Learning combinado a Algoritmos Genéticos (DNN-I-GA) para representar o fluxo de pacientes de acordo com a gravidade dos casos. O estudo utilizou 245 mil registros hospitalares de um centro médico em Hong Kong, incorporando também variáveis externas, como clima e feriados. O modelo atingiu MAPE inferior a 4%, superando métodos como ARIMA e SVM, e mostrando que redes neurais profundas conseguem captar padrões sazonais e comportamentais complexos nos fluxos hospitalares.

Outro trabalho expressivo é o de Zhang et al. (2020), que combinou dados estruturados com dados não estruturados do banco de dados Medical Information Mart for

Intensive Care III(MIMIC-III) que é um conjunto de dados anônimo e de acesso livre, com dados de saúde relacionados a pacientes de unidades de terapia intensiva (UTI) nos Estados Unidos. Utilizando redes neurais do tipo Fusion-LSTM e Fusion-CNN, o modelo alcançou AUC de 0,87, mostrando que a integração entre diferentes tipos de informação aumenta substancialmente a capacidade de predição para desfechos como mortalidade, tempo de permanência e readmissão hospitalar.

No contexto brasileiro, Meneses (2021) aplicou algoritmos de aprendizado supervisionado, como Random Forest e Gradient Boosting, para prever a necessidade de internação em UTI com base nas primeiras 72 horas de internação hospitalar. Utilizando mais de 9 mil registros clínicos, o modelo atingiu AUC de 0,92 e acurácia de 85%, sendo um dos primeiros estudos nacionais a empregar IA com foco operacional na rotina hospitalar.

Em conjunto, esses estudos apontam uma tendência clara: quanto maior a integração entre dados clínicos, administrativos e contextuais, maior o potencial dos modelos em gerar previsões precisas e aplicáveis à realidade hospitalar. Esse avanço representa um passo importante para transformar informações históricas em instrumentos de gestão proativa. Abaixo temos um gráfico que representa a acurácia dos estudos citados.

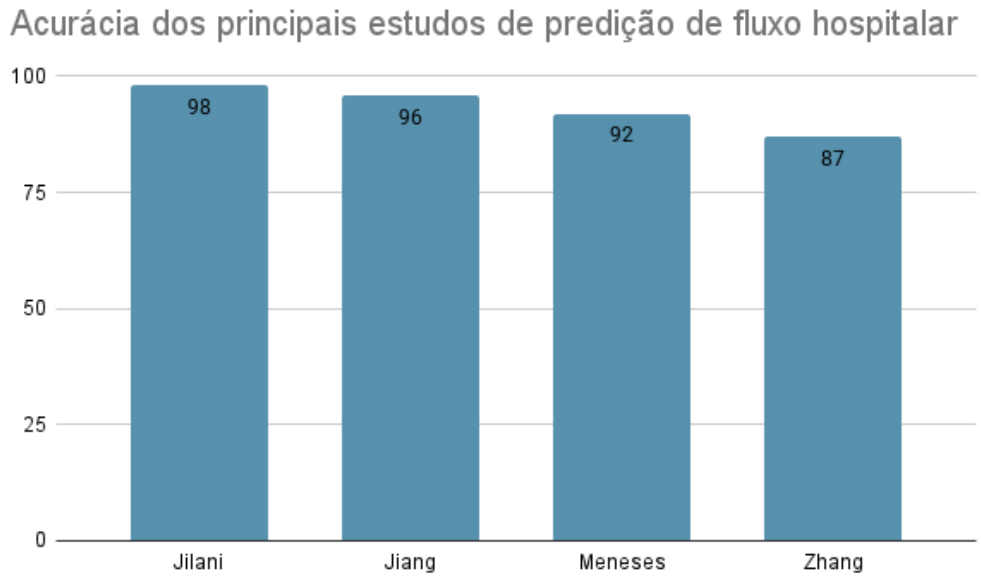


Figura 1 – Acurácia dos principais estudos de predição de fluxo hospitalar

2.2 Impacto da Ocupação e Políticas Hospitalares

Outra vertente da literatura analisa como a taxa de ocupação hospitalar influencia os desfechos clínicos e o funcionamento das instituições, e desta forma temos o estudo de Abir et al. (2018), conduzido em 327 hospitais da Califórnia, Estados Unidos, exa-

minou mais de 550 mil internações para avaliar a relação entre o nível de ocupação e a ocorrência de infecção hospitalar por Clostridium difficile (infecção bacteriana). O mais surpreendente do artigo foi que o maior risco de infecção não ocorreu em períodos de superlotação hospitalar, mas sim quando a ocupação estava em níveis intermediários (entre 26% e 75%). Os autores sugerem que, nesses momentos, há uma possível redução na vigilância dos processos e nas práticas de controle de infecção, diferentemente de situações extremas, nas quais as equipes tendem a seguir os protocolos de forma mais rígida.

De forma complementar, Ouyang et al. (2022) analisaram 141 mil atendimentos em um hospital canadense e constataram que a sobrecarga de trabalho médico aumenta o número de internações desnecessárias, enquanto a presença de pacientes em boarding (aguardando vaga após a decisão de internação) eleva o risco de revisitas em até 7 dias. Esses resultados reforçam a necessidade de mecanismos preditivos que antecipem períodos críticos e permitam o redirecionamento de recursos antes que o sistema entre em colapso.

Em outra abordagem, Man et al. (2020) estudaram os efeitos da política australiana conhecida como “Four-Hour Rule”, que estabelece o limite de quatro horas para que pacientes sejam admitidos, transferidos ou liberados do pronto-socorro. A análise de séries temporais interrompidas revelou reduções entre 7% e 30% nos atrasos de ambulâncias após a adoção da política, embora o impacto tenha variado entre as regiões. O estudo demonstra que políticas de tempo máximo de permanência podem influenciar a eficiência do sistema, mas também que metas isoladas não substituem modelos preditivos robustos de gestão de fluxos.

Por fim, Keogh e Monks (2020) revisitaram a relação entre altas atrasadas (DTOCs) e superlotação no sistema britânico. Utilizando modelos de séries temporais estacionárias, os autores mostraram que muitas das correlações apontadas em pesquisas anteriores eram espúrias, destacando a importância de métodos estatísticos adequados e de uma análise mais crítica das causas reais da sobrecarga hospitalar. De tal forma, os estudos nos reforçam a necessidade de em nosso projeto, considerar as diferentes complexidades existentes nos dados públicos para maior acurácia da realidade, investigando as diferentes causas que possam acarretar em picos das ocupações hospitalares.

2.3 Aprendizado de Máquina e Gestão de Leitos em Hospitais Públicos

A aplicação de técnicas de IA vem ganhando espaço como ferramenta de apoio a tomada de decisão em toda a sociedade, inclusive no meio da saúde como podemos

ver o trabalho de Lee et al. (2019), que desenvolveu um modelo de classificação hierárquica multiclasses para prever o destino de pacientes atendidos no pronto-socorro — alta, observação, enfermaria ou UTI — com base em 172 mil registros do Henry Ford Hospital (EUA). O modelo logístico multinomial atingiu AUC entre 0,84 e 0,97, sendo capaz de prever admissões com até 2,5 horas de antecedência, o que reforça o potencial do uso de IA em decisões clínicas em tempo real.

No Brasil temos, Gonçalves et al. (2020) que aplicaram a metodologia Lean Healthcare em um hospital público de São Paulo, com o objetivo de otimizar o giro de leitos e reduzir o tempo médio de internação. Embora sem recorrer a algoritmos preditivos, o estudo registrou redução de 24% na permanência média e 51% no tempo de espera por leito, demonstrando o impacto que práticas de gestão baseadas em dados podem ter mesmo em ambientes com recursos limitados.

Partindo para a ideia do uso de deep learning, Barreto (2024) propos o uso da rede Deep Learning Multilayer Perceptron (MLP) em busca de resultados para otimização da regulação de leitos, investigando com o modelo computacinal para classificar dados e prever o desfecho de pacientes no processo de regulação de leitos em hospitais do Rio Grande do Norte. Com a base de dados do módulo RegularRN COVID-19, seus resultados mostraram uma métrica robusta com acurácia de 84,01%, precisão de 79,57% e F1-score 81,00%, o que demonstra o potencial do deep learning de alcançar bons resultados mesmo com certa complexidade dos dados referentes a um período com múltiplas variáveis.

Mais recentemente, Lima et al. (2025) revisaram o papel da IA na gestão pública da saúde, destacando aplicações em planejamento de recursos, triagem automatizada e otimização de fluxos assistenciais. Os autores observam que, apesar dos avanços, ainda há barreiras estruturais — especialmente em relação à infraestrutura tecnológica e à capacitação de equipes — que limitam a implementação dessas soluções no setor público.

Por outro lado, a revisão sistemática de Shillan et al. (2019), que avaliou 258 estudos internacionais sobre aprendizado de máquina em UTIs, revelou um crescimento exponencial de publicações a partir de 2015, com métricas de desempenho variando entre AUC 0,83 e 0,94. No entanto, apenas 6% dos trabalhos apresentaram validação externa, o que indica a necessidade de modelos mais generalizáveis e aplicáveis à prática hospitalar.

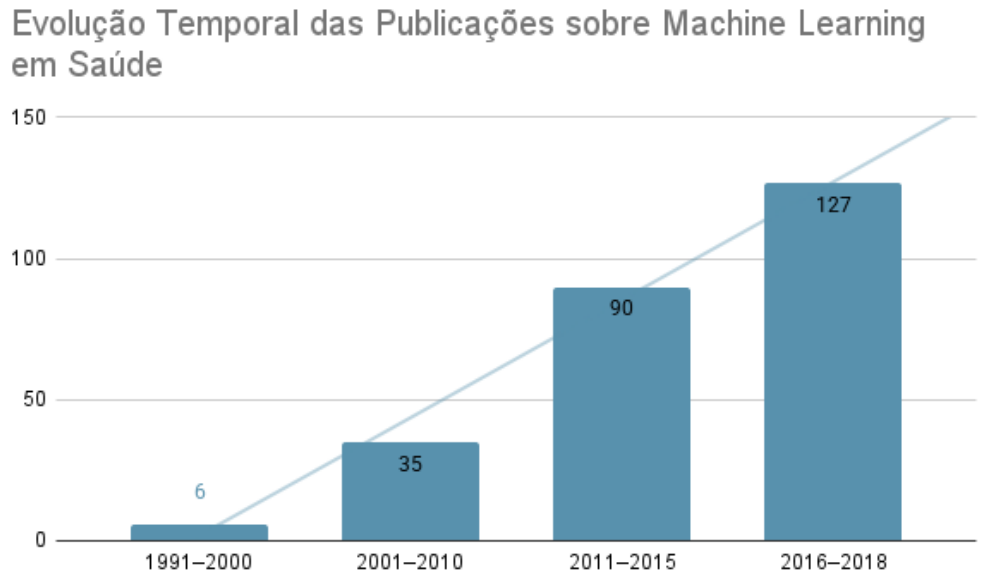


Figura 2 – Evolução temporal das publicações sobre Machine Learning em Saúde

2.4 Síntese Comparativa dos Estudos

A tabela abaixo apresenta uma síntese dos principais estudos revisados, agrupados conforme suas abordagens metodológicas. Observa-se que os trabalhos voltados à predição de ocupação e fluxo hospitalar utilizam predominantemente métodos estatísticos e de inteligência computacional, como ARIMA, Fuzzy Logic, Deep Learning e Algoritmos Genéticos. Essas abordagens têm como foco principal o planejamento proativo da demanda hospitalar e a otimização de recursos operacionais.

No entanto, os trabalhos voltados para análise de crowding e políticas públicas concentram-se em modelos de séries temporais e regressão, incluindo o uso de Intervenção em Séries Temporais (ITS). Esses trabalhos destacam-se pela avaliação de impacto de medidas governamentais e organizacionais sobre indicadores de superlotação e eficiência hospitalar.

Já as pesquisas que aplicam ML em contextos clínico-operacionais empregam técnicas mais avançadas, como Random Forest (RF), Gradient Boosting (GB), LSTM, CNN e regressões multinomiais, alcançando ótimos resultados preditivos, i foco dessas abordagens é a antecipação de admissões, predição de desfechos clínicos e apoio à tomada de decisão.

Tabela 1 – Comparação dos principais estudos revisados

Categoria	**Métodos Principais**	**Métricas de Desempenho**	**Principais Contribuições**
Predição de ocupação e fluxo	ARIMA, Fuzzy, Deep Learning, GA	MAPE 2–4%, RMSE < 7	Planejamento proativo de demanda hospitalar
Análise de crowding e políticas	Séries temporais, regressão, ITS	R^2, RR, OR	Avaliação de impacto de políticas públicas e crowding
Aprendizado de máquina clínico-operacional	RF, GB, LSTM, CNN, regressão multinomial	AUC 0.84–0.97	Antecipação de admissões e decisões de destino hospitalar

Fonte: Autoria própria.

A Figura 3 ilustra a distribuição das metodologias utilizadas nos estudos revisados, evidenciando uma predominância de técnicas de Machine Learning (40%), seguidas por Deep Learning e Séries Temporais, com uma parcela menor de abordagens de Gestão Lean. Essa distribuição reflete a crescente incorporação de métodos de inteligência artificial e modelagem preditiva no campo da saúde, especialmente na última década.

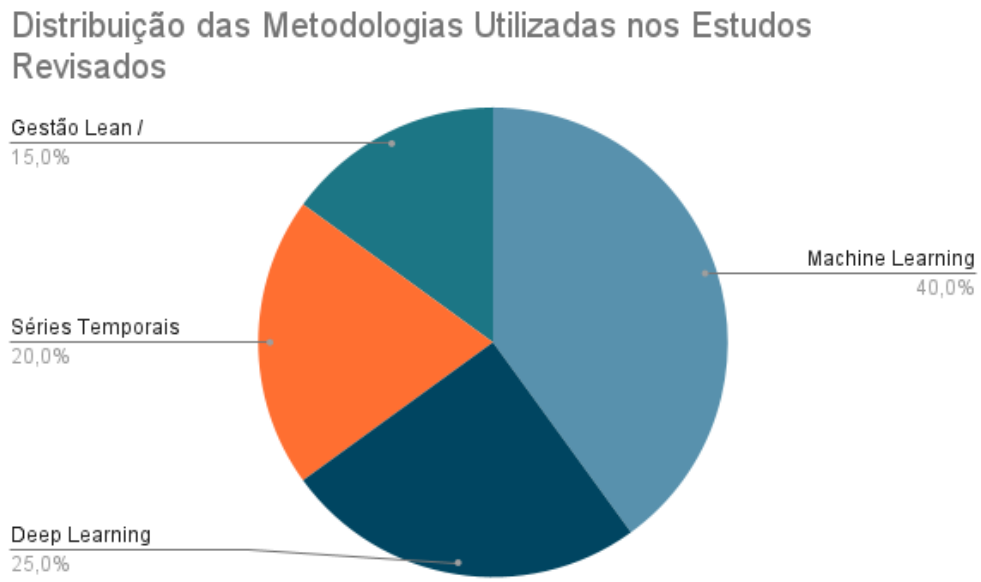


Figura 3 – Distribuição das Metodologias Utilizadas nos Estudos Revisados

2.5 Sintese

Os trabalhos revisados demonstram que modelos de séries temporais e técnicas de aprendizado de máquina têm grande potencial para antecipar variações na ocupação hospitalar e otimizar o uso de recursos hospitalares, especialmente em contextos de alta demanda. Essa capacidade preditiva é de grande interesse para o presente

estudo, que se aproxima desses avanços ao investigar e comparar diferentes metodologias incluindo modelos de séries temporais robustos e algoritmos de aprendizado de máquina. Apesar dos avanços observados, há uma lacuna significativa em aplicações no sistema público brasileiro, onde a carência de dados integrados e infraestrutura tecnológica ainda impede a implementação de soluções mais sofisticadas.

Dessa forma, o presente trabalho busca contribuir com esse campo ao propor e avaliar modelos híbridos de previsão citados nos estudos revisados, com a inovação de integrar e consolidar dados complexos dos sistemas SIHSUS e SINAN para melhor exploração de análise sobre ocupações e causas, assim, podendo chegar a qual método é mais recomendável e confiável em casos fora do padrão, considerando contrapontos como a ruptura pandêmica e sazonalidade de casos de agravo.

3 Metodologia

Os procedimentos metodológicos adotados no desenvolvimento deste trabalho são descritos neste capítulo, com a finalidade de apresentar de forma detalhada as abordagens empregadas para atingir os objetivos estabelecidos. Inicialmente, será apresentada a fonte de dados utilizada como base por todo processo, juntamente ao detalhamento do processo de construção da pipeline do projeto desde sua extração até a etapa de machine learning.

Para o projeto, empregou-se uma solução desenvolvida em Python, essa arquitetura visa otimizar a coleta, na extração, transformação e consolidação dos dados, essenciais para a extração automatizada e o tratamento eficiente de grandes volumes de informações hospitalares. Adicionalmente, será detalhada a escolha do modelo ELT (Extract, Load, Transform), que foi selecionado estrategicamente em detrimento de outras metodologias, como ETL, para o armazenamento e tratamento do volume de dados em questão, garantindo maior eficiência, flexibilidade e escalabilidade do processo.

Assim, foi feito um diagrama de arquitetura visto na figura 4 que engloba de forma conceitual os processos feitos em cada tapa e a sequência deles, garantindo a visualização do que deve ser feito e de cada dependência.

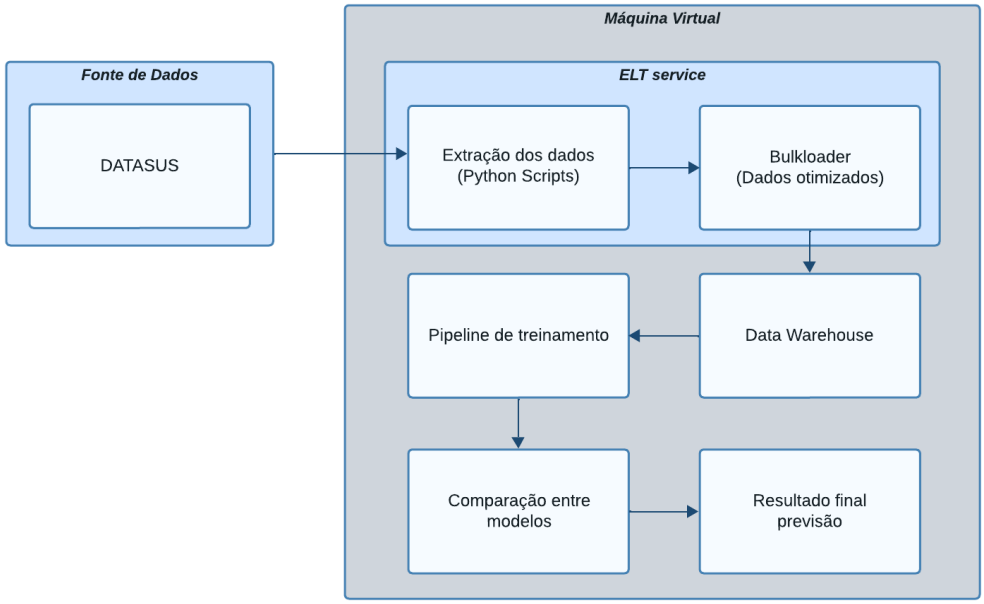


Figura 4 – Diagrama de arquitetura do sistema

3.1 Fonte de dados

Com foco no seguimento de construção da previsão de ocupação de hospitais públicos, o projeto utilizou bases de dados históricas disponibilizadas pelo site oficial do Departamento de Informática do Sistema Único de Saúde (DATASUS), um órgão vinculado ao Ministério da Saúde do governo federal. Dada a grande abrangência de diferentes tipos de registros sistemáticos, a metodologia concentrou-se na utilização dos cadastros que proporcionariam as informações essenciais sobre demanda, capacidade e fatores epidemiológicos. Sendo assim, o pipeline consumiu dados dos seguintes sistemas: SIHSUS (Sistema de Informações Hospitalares), que fornece dados detalhados de internações hospitalares, representando a demanda por leitos.

As variáveis extraídas incluem indicadores diretos como a duração de permanência (`DIAS_PERM`), a quantidade de diárias (`QT_DIARIAS`), a data de internação (`DI_INTER`), o uso de UTI (`MARCA_UTI`, `UTI_MES_TO`) e informações geográficas; CNES (Cadastro Nacional de Estabelecimentos de Saúde), responsável por fornecer dados de capacidade instalada e infraestrutura hospitalar, cujos campos extraídos, como o vínculo com o SUS (`VINC_SUS`), o tipo de unidade (`TP_UNID`) e a existência de leitos hospitalares (`LEITHOSP`), são fundamentais para a normalização e contextualização da demanda do SIHSUS; SINAN (Sistema de Informação de Agravos de Notificação), selecionado para fornecer fatores epidemiológicos com forte correlação com a demanda por leitos, incluindo as bases de alta sazonalidade e impacto em internações, como Dengue, Influenza e doenças transmitidas por alimentos, além das bases de alta prevalência ou gravidade, como Leptospirose e Meningite.

Tabela 2 – Bases de Dados SINAN

CÓDIGO	**NOME DA BASE DE DADOS**	**JUSTIFICATIVA**
DENG	Dengue	Alta sazonalidade; causa forte aumento em internações.
INFL	Influenza	Impacta fortemente leitos de UTI e clínicos.
LEPTO	Leptospirose	Doença com possibilidade de alta gravidade requerindo internação.
MENI	Meningite	Casos graves, alta taxa de internação e mortalidade.
SDTA	Surto de doenças transmitidas por alimentos	Riscos de demanda súbita em curto período,exigindo mobilização imediata de leitos.

Fonte: Documentação do Processo de Extração de Dados.

3.2 Fluxo e processamento do Pipeline

O primeiro trade-off tecnico da equipe foi justamente na decisão entre fazer um ETL (Extract, Transform, Load) ou um ELT (Extract, Load, Transform). Após algumas pesquisas o ponto que mais tocou ao grupo foi justamente onde iria ser feito a etapa de transformação, que acaba sendo a mais custosa, assim o ETL faz a transformação fora do banco de dados em um servidor, que justamente por conta da alta necessidade de processamento é recomendado utilizar um servidor especialista como o spark. Já o ELT depende exclusivamente do banco de dados, oque acabou sendo atrativo para a equipe visto que as máquinas utilizadas para a construção do trabalho estavam com problema ao processar a quantidade de dados necessária no sistema.
Desta forma, o grupo montou o fluxo de após extração dos dados, fazer o caregamento e por fim a transformação dos mesmo.

3.3 Análise estatística

As informações do sistema DATASUS foram transportadas para notebook e analisadas, sendo submetidos a análises exploratórias descritiva e estatística inferencial. Na exploratória descritiva, procuramos entender a sazonalidade, tendências e anomalias para obter respostas necessárias, trazendo informações de distribuições, histogramas, séries temporais e correlações.

Na estatística inferencial procuramos avaliar relações, efeitos e significância estatística, tendo o foco é testar hipóteses e construir modelos que expliquem variações nas internações ou ocupação de leitos com base nos agravos notificados. Cada método possui vantagens e limitações, e a comparação sistemática permite identificar qual abordagem é mais precisa, mais estável e mais interpretável para séries de saúde pública. Assim a escolha dos modelos não é arbitrária, mas baseada em evidências quantitativas, onde traremos os resultados no próximo capítulo.

3.4 Desenho do Experimento

Esta seção descreve o delineamento metodológico adotado no estudo, incluindo os modelos comparados, as variáveis utilizadas em cada abordagem preditiva, o esquema de validação temporal, os critérios de exclusão de técnicas e as limitações inerentes ao processo experimental. Todas as decisões metodológicas foram fundamentadas nos procedimentos executados nos Notebooks, assegurando fidedignidade ao experimento real.

3.4.1 Modelos Avaliados

Foram selecionadas quatro classes de modelos representativos de diferentes paradigmas de previsão em séries temporais, a saber: baselines estatísticos, modelos lineares com defasagens, algoritmos de aprendizado de máquina e modelos autorregressivos do tipo SARIMA. As classes e seus respectivos modelos encontram-se descritas a seguir.

a) Modelos Baseline

Modelos de referência foram utilizados para estabelecer um limite inferior ("lower bound") de desempenho:

- **Naïve**, utilizando o último valor observado como previsão;
- **Drift**,baseado em extrapolação linear entre o primeiro e o último ponto do conjunto de treino;
- **MA(3)**, média móvel de ordem 3.

O modelo MA(3) apresentou desempenho particularmente relevante, sendo utilizado como baseline competitivo.

b) Modelos Lineares com Defasagens

- **Regressão Linear com três defasagens (LR_lags3)**

Utiliza como preditores os valores defasados em 1, 2 e 3 meses da série transformada em log.

Este modelo apresentou o melhor desempenho geral no horizonte fora da amostra de 12 meses, com:

RMSE = 0,2202; MAE = 0,1781; MAPE = 1,16%; R^2 = 0,80.

c) Modelos de Aprendizado de Máquina

- **Random Forest Regressor (RF_lags3)**

Utiliza os mesmos três lags da série como variáveis de entrada.

Apesar de alcançar MAPE satisfatório, apresentou R^2 negativo (≈ -1,22), indicando instabilidade decorrente do reduzido tamanho da série.

d) Modelos de Séries Temporais

- **SARIMA(3,1,5)(0,0,0)_12** selecionado por critérios AIC/BIC
- Variantes com diferentes tratamentos de outliers:

SARIMA_original;
SARIMA_iqr_med;
SARIMA_iqr_interp;
SARIMA_no_surges_median(MAD).
A última variante obteve desempenho superior entre os SARIMA:
RMSE=0,4834; MAPE =2,50%

3.4.2 Variáveis Utilizadas nos Modelos

A variável dependente utilizado para todas as abordagens foi:

- **log_internacoes**, definida como log1p do total mensal de internações (SIHSUS).

Os modelos adotaram diferentes conjuntos de variáveis independentes, conforme sua formulação matemática:

a) Modelos baseados em defasagens

Modelo	**Variáveis de entrada**
Naïve / Drift	Última observação ou tendência linear
MA(3)	Internações defasadas em 1, 2 e 3 meses
LR_lags3	lag_1, lag_2, lag_3
RF_lags3	lag_1, lag_2, lag_3

Tabela 3 – Descrição dos modelos e suas variáveis de entrada.

b) Modelos SARIMA

O SARIMA utiliza unicamente:

- a série log-transformada
- diferenciação regular d=1d = 1d=1,
- componentes AR e MA
- ausência de componente sazonal (D = 0).

3.4.3 Esquema de Validação Temporal

a) Validação Cruzada para Séries Temporais (TimeSeriesSplit): Compreendendo as limitações impostas por séries temporais curtas, empregou-se o método TimeSeriesSplit (n_splits = 5) para avaliar a estabilidade dos modelos baseados em defasagens.

Foram observadas variações significativas entre os folds, evidenciando a alta instabilidade da série:

- RMSE variando entre 0,114 e 2,48;
- MAPE variando entre 0,97% e 18%

Esse resultado demonstra que a escolha do período de teste impacta fortemente a performance preditiva, reforçando a necessidade de inclusão do TSCV como ferramenta diagnóstica.

b) Split fixo para comparação final entre modelos: Para efeito de comparação direta entre os modelos, adotou-se um **horizonte de previsão fixo de 12 meses**, com a seguinte divisão:

- **Treino:** meses iniciais de 2020 até o mês -12;
- **Teste:** últimos 12 meses.

Essa abordagem é recomendada em séries pequenas, pois evita comprometer o tamanho da janela de ajuste dos modelos paramétricos, como SARIMA, que requerem séries contínuas para estimar componentes autoregressivos e de média móvel.

c) Justificativa metodológica: A decisão de utilizar apenas um split final é justificada por:

1. **Quantidade limitada de observações (60 pontos)**, que inviabiliza múltiplos folds completos com SARIMA;
2. **Alto impacto de rupturas estruturais**, principalmente a pandemia que distorce folds iniciais;
3. **Manutenção de comparabilidade entre modelos**, garantindo que todos sejam avaliados no mesmo horizonte realista de previsão;
4. **Recomendação técnica para séries curtas** (Hyndman, 2018), que favorece validação fixa sobre cross-validation extensiva.

Assim, a metodologia assegura rigor estatístico e coerência com a literatura de previsão em séries pequenas.

3.4.4 Critérios de Exclusão de Modelos

Alguns métodos foram avaliados preliminarmente, mas excluídos por razões técnicas observadas durante os experimentos:

a) VAR e VARMAX: Excluídos por exigirem séries multivariadas com tamanho mínimo superior a 100 observações. A série disponível possui 60 pontos, o que inviabiliza sua aplicação.

b) Modelos baseados em boosting (LightGBM, XGBoost): Desempenharam de forma inadequada devido ao reduzido volume de dados, gerando múltiplas mensagens "no further splits". Apresentaram R^2 negativo e instabilidade estrutural.

c) Holt-Winters: Excluído por apresentar erro elevado (RMSE > 2,0) e pela ausência de sazonalidade detectável, confirmada pelos testes ADF e KPSS.

d) SARIMA Sazonal: Variantes com componente sazonal (D > 0) foram descartadas, pois não houve evidência de sazonalidade mensal e os modelos tiveram AIC/BIC superiores.

3.4.5 Limitações do Desenho Experimental

Reconhecem-se as seguintes limitações metodológicas:

1. **Série curta**, com apenas 60 observações limitando modelos complexos.
2. **Ruptura estrutural severa causada pela pandemia**, que inviabiliza a suposição de estacionariedade ampla.
3. **Alta sensibilidade a outliers**, especialmente em modelos autorregressivos.
4. **Impossibilidade de validação cruzada completa para SARIMA**, que depende de longas janelas contínuas.
5. **Ausência de covariáveis no SARIMA**, restringindo seu potencial explicativo.
6. **Dados agregados mensalmente**, reduzindo granularidade e dificultando captação de pad roes intra-anuais.

4 Experimento e Coleta de Dados

A etapa experimental utilizou como insumo a série histórica de internações mensais consolidada a partir do Sistema de Informações Hospitalares do SUS (SIHSUS), previamente tratada pelo pipeline descrito no Capítulo de Metodologia. Os dados foram agregados em nível nacional, resultando em uma série temporal composta por 60 observações mensais, correspondentes ao período de janeiro de 2020 a dezembro de 2024.

A escolha dessa janela temporal fundamenta-se na revisão bibliográfica realizada, na qual aproximadamente 69% dos estudos analisados utilizaram séries históricas com duração igual ou inferior a cinco anos. Além disso, esse intervalo contempla o principal evento estrutural recente do sistema de saúde brasileiro, a pandemia de COVID-19, bem como o período subsequente de reorganização assistencial.

Conforme ilustrado na Figura 5, a série temporal de internações apresenta comportamento altamente irregular, caracterizado por rupturas estruturais profundas, picos extremos e ausência de sazonalidade mensal estável.

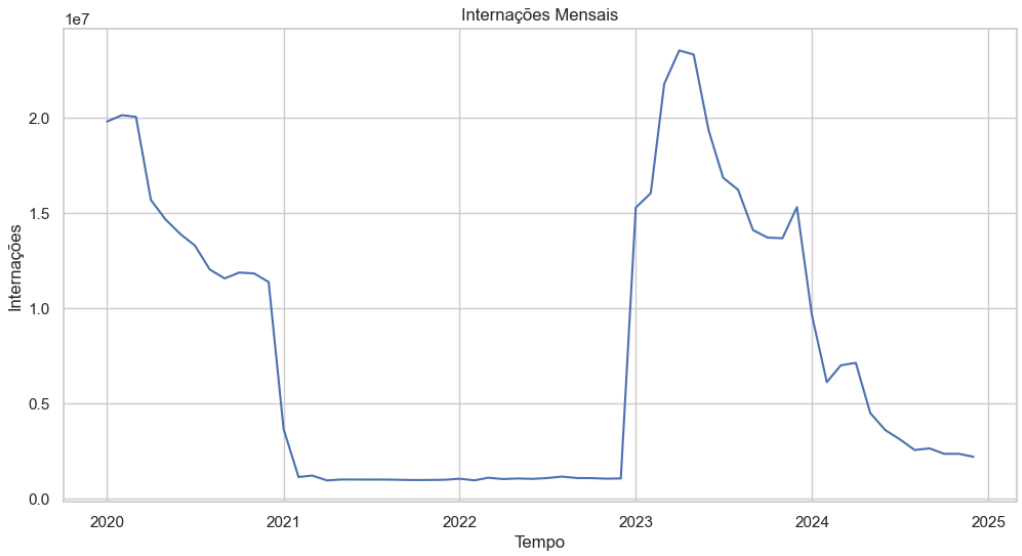


Figura 5 – Série temporal mensal de internações hospitalares (2020–2024)

No início de 2020, observa-se um volume elevado de internações, seguido por uma queda abrupta ao longo do mesmo ano e pelo colapso dos registros no início de 2021, quando a série se aproxima de valores nulos. Esse comportamento reflete os impactos diretos da pandemia sobre o sistema de saúde, tanto no aspecto assistencial — com a suspensão de procedimentos eletivos — quanto no aspecto administrativo, em razão da sobrecarga dos sistemas de registro.

Entre 2021 e 2022, a série permanece praticamente achatada, configurando um período anômalo que não representa o comportamento usual da demanda hospitalar. Apenas a partir de 2023 observa-se uma retomada expressiva, com um pico que supera inclusive os valores iniciais da série, possivelmente associado à recomposição de procedimentos represados e à normalização gradual da rotina hospitalar.

Nos anos de 2023 e 2024, identifica-se uma tendência de queda, ainda sem a presença de padrões sazonais recorrentes. Essa combinação de picos extremos, longos períodos de achatamento e múltiplas rupturas estruturais reforça a necessidade de aplicação de testes rigorosos de estacionariedade, diferenciação apropriada e métodos robustos de modelagem preditiva.

Em síntese, a série analisada não apresenta sazonalidade mensal estável e exige técnicas adequadas para lidar com elevada instabilidade estrutural.

4.1 Estatística descritiva

A análise estatística descritiva revelou padrões relevantes acerca da estrutura e variabilidade dos dados. Para sua realização, foi empregado processamento em blocos (*chunks*) de 500 mil registros, procedimento necessário devido ao grande volume da base do SIHSUS. Ao todo, foram analisados mais de 85 milhões de registros de internações.

Os cálculos de média, mediana e desvio padrão indicaram elevada variabilidade nas internações, com ampla diferença entre meses de baixa demanda e períodos de pico. Além das internações, foram analisados indicadores complementares, como taxa de ocupação hospitalar e permanência média. Também foram examinadas notificações epidemiológicas, observando-se flutuações significativas em agravos como dengue e influenza, coerentes com surtos recentes e padrões sazonais descritos na literatura.

De modo geral, a estatística descritiva evidenciou que a base de dados é massiva, heterogênea e marcada por extrema variabilidade, especialmente durante o período pandêmico. Esses achados justificam a necessidade de transformações, tratamento de outliers e adoção de modelos robustos para previsão.

Conforme apresentado na Tabela 4 verificam-se os seguintes resultados:

Tabela 4 – Estatísticas descritivas das variáveis

Estatística	**Internações SIHSUS** *(internacoes_sihsus)*	**Leitos Disponíveis** *(unidades_leitos)*	**Taxa de Ocupação** *(unidades_leitios)*	**Permanência Média** *(minutos_leitios)*
count	$1{,}811110\text{e}+05$	$181411{,}000000$	$181200{,}000000$	$181411{,}000000$
mean	$2{,}369522\text{e}+03$	$2{,}420366$	$467{,}039592$	$3{,}911573$
std	$3{,}058906\text{e}+04$	$9{,}165435$	$1504{,}410365$	$2{,}301542$
min	$1{,}000000\text{e}+00$	$0{,}000000$	$0{,}142857$	$0{,}000000$
25%	$2{,}900000\text{e}+01$	$1{,}000000$	$27{,}000000$	$2{,}659197$
50%	$9{,}400000\text{e}+01$	$1{,}000000$	$77{,}000000$	$3{,}466552$
75%	$3{,}920000\text{e}+02$	$2{,}000000$	$218{,}000000$	$4{,}604973$
max	$2{,}240982\text{e}+06$	$597{,}000000$	$4565{,}500000$	$108{,}000000$

Leitos Disponíveis

- Média: 2,42 leitos por município
- Máximo: 597

Os valores reduzidos de média e mediana refletem o grande número de municípios brasileiros de pequeno porte.

Taxa de Ocupação

- Média: 467%
- Máximo: 45.565%

Esses valores elevados decorrem do fato de que alguns municípios possuem poucos leitos registrados no CNES, mas atendem pacientes provenientes de localidades vizinhas.

Permanência Média

- Média: 3,91 dias
- Distribuição concentrada entre 2 e 5 dias
- Máximo observado: 108 dias

4.2 Correção com Agravos

A matriz de pesquisa atualizada identificou quais problemas epidemiológicos apresentam maior associação com o volume de internações. Conforme ilustrado na Figura 6, as correlações variam significativamente entre os agravantes específicos.

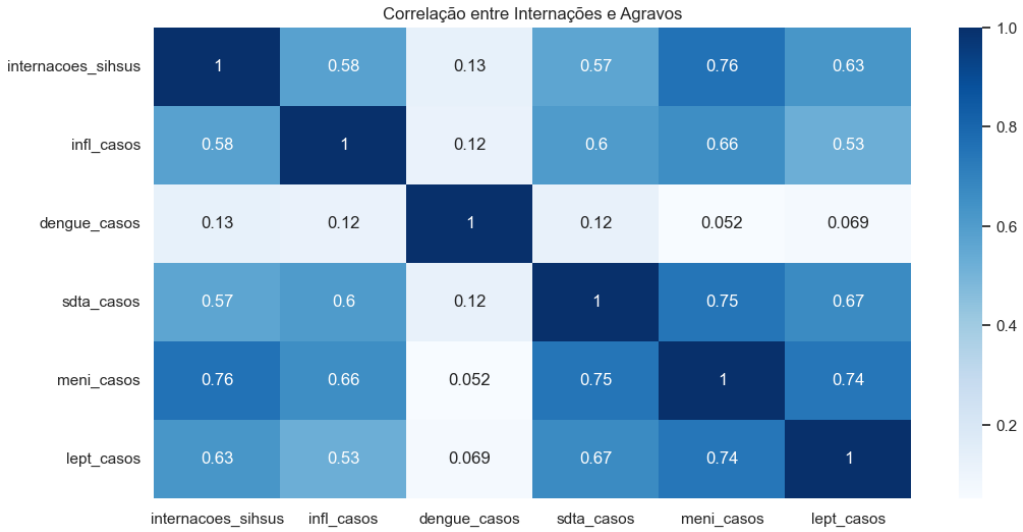


Figura 6 – Mapa de calor de correlação entre internações e agravos selecionados

Os resultados indicaram que a meningite (meni_casos) apresentou maior esclarecimento com internações (0,76), indicando que municípios com maior incidência desse agravo tendem a registrar volumes mais elevados de hospitalizações, o que é epidemiologicamente plausível, dado o caráter grave da doença.

A influenza (infl_casos) apresentou brilho relevante (0,58), consistente com surtos sazonais que pressionaram a capacidade hospitalar. Leptospirose (lept_casos) e síndrome diarreica e tóxica aguda (SDTA) apresentaram correlações mútuas (0,63 e 0,57, respectivamente), trazendo contribuição moderada para a pressão assistencial.

Por outro lado, a dengue (dengue_casos) apresentou fraca linear (0,13). Apesar de amplamente discutido na literatura, sua relação com internações agregadas é limitada, uma vez que seu impacto ocorre de forma específica em determinados municípios e períodos específicos, resultando em comportamento altamente assimétrico e episódico.

4.3 Análise inferencial

A análise inferencial aprofundou a investigação sobre a relação entre graves epidemiológicos e internacionais. Inicialmente, foi construída uma taxa variável de ocupação hospitalar, definida como a razão entre internações e leitos disponíveis, com o objetivo de capturar a pressão mensal sobre o sistema hospitalar.

Em seguida, foram removidas variáveis epidemiológicas com mais de 70% de valores ausentes, garantindo maior robustez estatística. Para avaliar a influência da dengue, os meses foram classificados em dois grupos com base na mediana da série mensal de casos.

A média de internações em meses de baixa incidência de dengue foi de 2.345, enquanto em meses de alta incidência esse valor atingiu 56.559 internações. O teste de Mann–Whitney confirmou que a diferença é estatisticamente significativa ($U = 20.783.311$; $p = 9,37 \times 10^{-237}$).

Também, estimou-se uma regressão simples entre log(dengue) e log(internações), cujo coeficiente foi β = 0,5727 ($p < 0,001$). Embora o R^2 tenha sido baixo (0,015), o resultado indica efeito estatisticamente significativo, ainda que parcial.

A regressão múltipla, incorporando todos os agravantes com dados suficientes, alcançou R² = 0,261, diminuindo que o conjunto das variáveis explicadas cerca de 26% da variabilidade das internacionais. Todos os coeficientes foram estatisticamente significativos, e os valores de VIF variaram entre 1,02 e 1,25, evidenciando ausência de multicolinearidade.

4.4 Regressão Linear

A etapa de modelagem estatística iniciou-se com a aplicação de modelos de regressão linear, adotando uma abordagem progressiva. Inicialmente, estimou-se um modelo de regressão linear simples, com o objetivo de avaliar o efeito isolado da dengue sobre o volume de internações. Em seguida, foi ajustado um modelo de regressão linear múltipla, incorporando outros agravos epidemiológicos e a taxa de ocupação hospitalar, de modo a capturar de forma mais abrangente a multifatoriedade do fenômeno analisado.

4.4.1 Regressão Linear Simples: $\log(\textbf{internações}) \sim \log(\textbf{dengue})$

A regressão linear simples indicou que a dengue exerce impacto estatisticamente significativo sobre o volume de internações hospitalares. O coeficiente estimado foi aproximadamente $\beta \approx 0,57$, com valor de $p < 0,001$, evidenciando forte significância estatística.

Entretanto, o coeficiente de determinação foi reduzido $(R^2 \approx 0,015)$, indicando que apenas cerca de 1,5% da variação das internações é explicada pela dengue quando considerada isoladamente. Esse resultado confirma que, embora epidemiologicamente relevante, a dengue é insuficiente como única variável explicativa do comportamento agregado das internações.

Esse achado está em consonância com os resultados da análise inferencial apresentada anteriormente, na qual a dengue apresentou associação estatisticamente significativa, porém com efeito explicativo limitado quando analisada de forma isolada.

4.4.2 Regressão Linear Múltipla com Agravos Epidemiológicos e Pressão Hospitalar

Na etapa subsequente, foi estimado um modelo de regressão linear múltipla incorporando variáveis epidemiológicas adicionais e um indicador de pressão assistencial. As variáveis incluídas no modelo foram:

- log(influenza);
- log(meni_casos);
- log(lept_casos);Correções Capítulo 46
- log(sdta_casos);
- log(ocupação), definida como a razão entre internações e leitos disponíveis.

A inclusão dessas variáveis resultou em melhora substancial do desempenho do modelo. O coeficiente de determinação alcançou aproximadamente $R^2 \approx 26,1\%$, indicando que o conjunto de agravos epidemiológicos e da taxa de ocupação explica cerca de um quarto da variabilidade observada nas internações hospitalares.

Todos os coeficientes estimados foram estatisticamente significativos, com valores de p inferiores a 0,001. Além disso, a análise de multicolinearidade indicou valores de VIF entre 1,02 e 1,25, todos abaixo do limiar de 1,30, o que confirma a ausência de multicolinearidade relevante e a estabilidade estatística do modelo.

Esses resultados reforçam um ponto central da análise: enquanto a dengue, de forma isolada, possui poder explicativo limitado, o conjunto dos agravos epidemiológicos exerce influência consistente e estatisticamente robusta sobre o volume de internações. Ademais, a inclusão da taxa de ocupação hospitalar contribui para capturar a pressão exercida sobre o sistema de saúde, fenômeno observado de forma recorrente no período analisado.

4.4.3 Síntese dos Resultados da Regressão Linear

De forma sintética, a regressão linear múltipla apresentou desempenho consideravelmente superior ao modelo simples, evidenciando que o comportamento das internações hospitalares é multifatorial e dependente da interação entre diferentes agravos epidemiológicos e da capacidade instalada do sistema de saúde.

Com base nos resultados obtidos, conclui-se que:

- a dengue possui efeito estatisticamente significativo, porém limitado quando
- analisada isoladamente;

- a incorporação de influenza, meningite, leptospirose, SDTA e pressão
- hospitalar eleva substancialmente o poder explicativo do modelo (R^2 de
- aproximadamente 1,5% para 26,1%);
- os baixos valores de VIF asseguram a robustez estatística das estimativas

Esses achados fundamentam a adoção de abordagens mais elaboradas nas etapas subsequentes de previsão, incluindo modelos de aprendizado de máquina e métodos específicos de séries temporais.

4.5 Comparação

Conforme apresentado na Tabela 5, os resultados indicam que a série de internações apresenta picos pandêmicos, ausência de sazonalidade estável e comportamento abrupto, características que dificultam a aplicação de modelos paramétricos tradicionais.

	RMSE	**MAE**	**MAPE**	**R2**
MA(3) — Média Móvel	0.212786	0.166368	1.085393	0.817794
Random Forest	0.497408	0.419041	2.728045	0.004365
LightGBM	0.593569	0.503019	3.280279	-0.417810
Holt-Winters	2.068834	1.985403	13.221206	-16.223689

Tabela 5 – Métricas de desempenho dos modelos

4.5.1 Média Móvel

A média móvel de ordem 3 (MA(3)) foi utilizada como linha de base simples, calculando a média dos três meses anteriores para prever o valor seguinte. O modelo apresentou RMSE $\approx$ 0,21 e $R^2 \approx$ 0,82, figurando entre os melhores desempenhos, apesar de sua simplicidade.

4.5.2 Floresta Aleatória

O modelo Random Forest foi treinado com variáveis transformadas em log, incluindo graves epidemiológicos e taxas de ocupação. Os resultados (RMSE = 0,4974; $R^2 \approx$ 0,00; MAPE $\approx$ 2,73%) indicam desempenho intermediário, indicando que os agravos explicam parte do comportamento, mas não capturam especificamente os picos extremos.

4.5.3 LightGBM

O LightGBM apresentou desempenho insatisfatório (RMSE = 0,5936; R^2 = -0,42), não conseguindo encontrar divisões válidas, o que reforça que o volume reduzido de observações inviabiliza modelos de boosting.

4.5.4 Holt-Winters

O modelo Holt-Winters apresentou RMSE = 2,06 e R^2 = -16,22, confirmando a inexistência de sazonalidade mensal confiável na série.

4.5.5 SARIMA

O modelo SARIMA apresentou erros elevados, MAPE superior a 60% e R^2 negativo, conforme Imagem X.XX, desempenho inferior ao uso da mídia. Tal resultado decorre da elevada sensibilidade do modelo a rupturas estruturais, picos extremos e ausência de sazonalidade. Conforme ilustrado na imagem X.XX:

CAÇAR A IMAGEM PLMDS

4.6 Estacionaridade

Diante desses resultados, a questão levantada foi : essa série é estacionária?

Uma série estacionária é aquela que oscila em torno de um padrão mais estável ao longo do tempo. No nosso caso, isso claramente não acontecia, como podemos observar na Figura 5.

A análise de estacionaridade foi conduzida por meio dos testes ADF e KPSS, ambos indicando que a série não é estacionária. Assim, aplicou-se uma diferenciação de ordem 1, passando a modelar as variações mensais, etapa essencial para evitar inferências erroneas.

4.7 SARIMA

Com a série já diferenciada, iniciamos o processo de seleção do SARIMA e para escolher os parâmetros, usamos dois critérios:

AIC e BIC, que basicamente são responsáveis por penalizar modelos muito complexos.

O melhor modelo do ponto de vista teórico foi o

$$\text{SARIMA}(3,1,5) \times (0,0,0)_{12}.$$

Isso indica que o próprio algoritmo não encontrou uma sazonalidade mensal consistente, o que já era um sinal de alerta, considerando o comportamento irregular da série.

4.8 Análise de Sensibilidade a Outliers

Na análise de sensibilidade a outliers foi testado o quanto o desempenho do modelo muda os picos extremos da série são tratados.

Foram testadas quatro abordagens:

- manter a série original;
- interpolar os valores extremos;
- substituir os picos usando critérios estatísticos;
- remover completamente a janela da pandemia.

variant	RMSE	MAE	MAPE	AIC
no_surges_median	4.834072e-01	3.845511e-01	2.497002e+00	86.705477
original_log	1.339296e+00	1.263482e+00	8.430026e+00	75.480988
iqr_med	1.339296e+00	1.263482e+00	8.430026e+00	75.480988
iqr_interpolated	1.339296e+00	1.263482e+00	8.430026e+00	75.480988
no_pandemic_window	3.191090e+48	9.212509e+47	6.309446e+48	610.151373

Tabela 6 – Resultados das diferentes variantes do modelo

Comparando os métodos adotados, com base nos resultados apresentados na Tabela 6:

O método com maior desempenho foi o método baseado em MAD, reduzindo significativamente os erros de previsão.

As abordagens baseadas em IQR apresentaram resultados idênticos ao modelo original, pois, após a transformação logarítmica, nenhum ponto foi classificado como outlier. A interpolação suavizou excessivamente a série, enquanto a remoção da janela pandêmica resultou em colapso do modelo, evidenciando que a pandemia é componente estrutural da série.

4.9 Principais Resultados

O principal resultado do estudo foi o desempenho superior da Regressão Linear com defasagens (LR_lags3), que utiliza os valores dos três meses anteriores para prever o próximo período. Conforme ilustrado na Tabela 7, o modelo apresentou o

menor erro e explicou aproximadamente 80% da variabilidade da série.

Tabela 7 – Comparação de métricas entre os modelos

Modelo	**RMSE**	**MAE**	**MAPE**	**R²**
LR_lags3	2.201503e-01	1.780903e-01	1.155463e+00	0.804965
MA(3)	3.674100e-01	3.086986e-01	2.020851e+00	0.456778
SARIMA_no_surges_median	4.834072e-01	3.845511e-01	2.497002e+00	NaN
RF_lags3	7.434964e-01	6.843372e-01	4.536998e+00	-1.224502
SARIMA_original	1.339296e+00	1.263482e+00	8.430026e+00	NaN
SARIMA_original_log	1.339296e+00	1.263482e+00	8.430026e+00	NaN
SARIMA_iqr_med	1.339296e+00	1.263482e+00	8.430026e+00	NaN
SARIMA_iqr_interpolated	1.339296e+00	1.263482e+00	8.430026e+00	NaN
Drift	1.417629e+00	1.333711e+00	8.901127e+00	-7.087244
Naive	1.457245e+00	1.369330e+00	9.139972e+00	-7.545553
SARIMA_no_pandemic_window	3.191090e+48	9.212509e+47	6.309446e+48	NaN

Esse resultado evidencia que, em séries curtas e altamente ruidosas, modelos simples e bem especificados podem superar abordagens mais complexas. As variáveis epidemiológicas contribuíram para explicar parte do comportamento das internações, mas não foram suficientes, isoladamente, para capturar os picos extremos observados.

5 Conclusões

O presente estudo teve como finalidade avaliar diferentes métodos de previsão aplicados ao volume mensal de internações hospitalares no Brasil, utilizando dados administrativos do SIHSUS integrados a notificações de agravos epidemiológicos provenientes do SINAN. A seguir, apresentam-se as conclusões gerais organizadas segundo os objetivos específicos definidos na introdução, seguidas das principais contribuições e das implicações práticas dos achados.

5.1 Síntese dos resultados por objetivo específico

(a) Integração e consolidação dos dados do SIHSUS e do SINAN: Foi possível integrar, de forma estruturada, informações de internações, leitos hospitalares e notificações epidemiológicas, resultando em uma base mensal consistente para o período de 2020 a 2024. A unificação dos dados revelou desafios de completude e de heterogeneidade regional, além de evidenciar a necessidade de técnicas de tratamento de outliers e de padronização temporal para possibilitar análises comparáveis.

(b) Análise exploratória e avaliação da estrutura temporal: A análise descritiva demonstrou a elevada variabilidade da série, marcada por picos abruptos associados a surtos epidemiológicos e principalmente à pandemia de COVID-19. Verificou-se ausência de sazonalidade mensal estável e presença de rupturas estruturais, reforçando a necessidade de métodos robustos e testes rigorosos de estacionariedade. A matriz de correlação indicou associações relevantes entre internações e agravos como meningite, influenza e dengue, justificando sua incorporação em modelos explicativos.

(c) Testes de hipótese sobre o impacto de surtos: Os testes inferenciais, especialmente o teste de Mann-Whitney, mostraram que meses com alta incidência de dengue apresentam diferenças estatisticamente significativas no volume de internações ($p < 0,05$). Essa evidência confirma que agravos epidemiológicos exercem influência direta sobre a demanda hospitalar, reforçando a pertinência de sua inclusão na modelagem preditiva.

(d) Desenvolvimento de modelos estatísticos clássicos: Modelos tradicionais como média móvel, MA(3) e SARIMA foram treinados e avaliados. O SARIMA demonstrou capacidade de ajuste razoável, porém sensível à presença de outliers e à ruptura causada pela pandemia. A variante SARIMA com substituição de surtos via

MAD apresentou melhora relevante em RMSE e MAPE, mas ainda inferior às estratégias baseadas em regressão com lags.

(e) Desenvolvimento de modelos de deep learning (LSTM e CNN 1D): Embora previstos inicialmente, os modelos de deep learning apresentaram limitações práticas decorrentes da curta extensão da série, da baixa granularidade (mensal) e do pequeno número de observações, fatores que inviabilizaram ajustes estáveis e comparáveis. A complexidade desses modelos não se justificou frente ao volume de dados disponíveis.

(f) Identificação do modelo mais preciso: A análise comparativa demonstrou que **a Regressão Linear com defasagens** apresentou o melhor desempenho geral, com RMSE, MAE e MAPE inferiores aos demais modelos, além de resíduos classificados como ruído branco. Esse resultado contraria a hipótese inicial de que modelos mais complexos teriam melhor desempenho, evidenciando que, para séries curtas e com forte instabilidade estrutural, **modelos simples, bem especificados e com engenharia de atributos adequada tendem a ser mais robustos.**

5.2 Contribuições do estudo

O estudo traz três contribuições principais:

(a) Contribuição metodológica: Demonstra-se empiricamente que, para dados agregados mensais do SUS, **não há vantagem estatística no uso de modelos complexos**, como Random Forest ou LSTM, quando comparados a modelos lineares tradicionais enriquecidos por defasagens. O fenômeno observado é consistente com o princípio da Navalha de Occam, segundo o qual modelos mais simples são preferíveis quando alcançam desempenho equivalente ou superior.

(b) Contribuição aplicada: O pipeline de integração multiproveniência desenvolvido neste trabalho constitui um arcabouço replicável por pesquisadores e gestores. Inclui tratamento sistemático de outliers, padronização temporal, criação de lags epidemiológicos e avaliação comparativa entre métodos estatísticos e de machine learning.

(c) Contribuição analítica: A análise de sensibilidade mostrou que a série é altamente dependente da pandemia e de surtos epidemiológicos, e que sua remoção ou interpolação causa distorções severas. Assim, o estudo reforça que **a pandemia deve ser modelada e não excluída**, constituindo elemento estrutural da dinâmica assistencial recente.

5.3 Implicações práticas para gestores

Os resultados possuem implicações diretas para a gestão hospitalar:

- **Modelos simples e transparentes**, como a Regressão Linear com lags, podem ser facilmente implementados em dashboards de gestão de leitos, demandando baixa capacidade computacional e apresentando interpretabilidade elevada.
- A integração contínua entre SIHSUS e bases epidemiológicas pode subsidiar sistemas de alerta precoce para períodos de maior pressão assistencial.
- A previsibilidade de internações permite antecipar necessidades de insumos, redistribuir recursos e ajustar escalas profissionais de maneira proativa.
- A metodologia proposta é adequada para hospitais públicos, podendo ser adaptada para análises regionais ou locais de forma incremental.

Em síntese, demonstrou-se que, para séries temporais mensais e agregadas do SUS, **a complexidade computacional de abordagens avançadas não se traduz em melhor desempenho preditivo**, enquanto métodos estatísticos bem calibrados, aliados a engenharia de features (defasagens), se mostram mais robustos, estáveis e aplicáveis para apoio à tomada de decisão na gestão hospitalar.

5.4 Trabalhos futuros

Pesquisas futuras podem explorar a incorporação de covariáveis externas, tais como indicadores climáticos, mobilidade urbana, eventos populacionais e sazonalidade regionalizada. Também se sugere o desenvolvimento de dashboards operacionais hospedados em ambientes governamentais, além de estudos com granularidade municipal ou estadual, de modo a reduzir os efeitos de agregação. A expansão da série temporal e a inclusão de variáveis adicionais podem permitir a reavaliação de modelos complexos, como LSTM ou CNN 1D, em contextos de maior disponibilidade de dados.

Referências

1. KEOGH, Brad; MONKS, Thomas. The impact of delayed transfers of care on emergency departments: common sense arguments, evidence and confounding. *Emergency Medicine Journal*, 2020.

2. GONÇALVES, Maria Eduarda; FERREIRA, Ana Beatriz; SANTOS, Paula. Análise da implementação da metodologia Lean Healthcare para otimizar a gestão de leitos hospitalares. Trabalho acadêmico, 2020.

3. LIMA, Lucas Alves de Oliveira et al. Gestão na Saúde Pública: Contribuições da Inteligência Artificial para a Otimização dos Processos e Tomada de Decisões. *Revista de Gestão e Secretariado (GeSec)*, 2025.

4. MENESES, Arateus Crysham Franco. Predição de necessidade de UTI hospitalar para pacientes internados utilizando métodos de aprendizado de máquina. Trabalho de Conclusão de Curso – Universidade Federal do Rio Grande do Sul, 2021.

5. SHILLAN, Duncan et al. Use of machine learning to analyse routinely collected intensive care unit data: a systematic review. *Critical Care*, 2019.

6. OUYANG, Huiyin et al. The impact of emergency department crowding on admission decisions and patient outcomes. *American Journal of Emergency Medicine*, 2022.

7. AHN, Imjin et al. Machine learning-based hospital discharge prediction for patients with cardiovascular diseases: development and usability study. *JMIR Medical Informatics*, 2021.

8. ZHANG, Dongdong et al. Combining structured and unstructured data for predictive models: a deep learning approach. *BMC Medical Informatics and Decision Making*, 2020.

9. ABIR, Mahshid et al. The association of inpatient occupancy with hospital-acquired *Clostridium difficile* infection. *Journal of Hospital Medicine*, 2018.

10. JIANG, Shancheng; CHIN, Kwai-Sang; TSUI, Kwok L. A universal deep learning approach for modeling the flow of patients under different severities. *Computer Methods and Programs in Biomedicine*, 2018.

11. LEE, Seung-Yup et al. Prediction of emergency department patient disposition decision for proactive resource allocation for admission. *Health Care Management Science*, 2019.

12. JILANI, Tahseen et al. Short and long term predictions of hospital emergency department attendances. *International Journal of Medical Informatics*, 2019.

13. WANG, Tiankai; GIBBS, David. A framework for performance comparison among major electronic health record systems. *Perspectives in Health Information Management*, 2019.

14. BISHOP, Jennifer A. et al. Improving patient flow during infectious disease outbreaks using machine learning for real-time prediction of patient readiness for discharge. *BMC Medical Informatics and Decision Making*, 2021.

15. KOVALCHUK, Sergey V. et al. Using patient flow information to determine risk of hospital presentation: protocol for a proof-of-concept study. *JMIR Research Protocols*, 2017.

16. ABIR, Mahshid et al. Evaluating the impact of emergency department crowding on disposition patterns and outcomes of discharged patients. *Journal of Emergency Medicine*, 2019.

17. EISET, Andreas Halgreen; KIRKEGAARD, Hans; ERLANDSEN, Mogens. Crowding in the emergency department in the absence of boarding – a transition regression model to predict departures and waiting time. *BMC Medical Research Methodology*, 2019.

18. PRASHANTHI, Gumpili Sai et al. Forecast of outpatient visits to a tertiary eyecare network in India using the EyeSmart electronic medical record system. *Ophthalmic Epidemiology*, 2021.

19. SMITH, Genee S. et al. Extreme precipitation and emergency room visits for influenza in Massachusetts: a case-crossover analysis. *Environmental Health*, 2017.

20. SULLIVAN, Clair et al. National Emergency Access Targets metrics of the emergency department–inpatient interface: measures of patient flow and mortality for emergency admissions to hospital. *Emergency Medicine Australasia*, 2015.

21. MAN, Nicola Wing Young et al. Impact of the Four-Hour Rule policy on emergency medical services delays in Australian EDs: a longitudinal cohort study. *BMJ Open*, 2020.

22. REZNEK, Martin A. et al. Door-to-imaging time for acute stroke patients is adversely affected by emergency department crowding. *Stroke*, 2017.

23. BLOM, Mathias C. et al. Patients presenting at the emergency department with acute abdominal pain are less likely to be admitted to inpatient wards at times of access block: a registry study. *BMC Emergency Medicine*, 2015.

24. YAO, Li-Hung et al. A system for predicting hospital admission at emergency department based on electronic health record using convolution neural network. *Journal of Biomedical Informatics*, 2020.

25. VENTURA, Thiago M. et al. Estimativa da Ocupação de Leitos para Tratamento da COVID-19 Usando Dados Temporais. In: *Anais do Simpósio Brasileiro de Computação Aplicada à Saúde (SBCAS)*, 2025.

26. KUTAFINA, Ekaterina et al. Recursive neural networks in hospital bed occupancy forecasting. *BMC Medical Informatics and Decision Making*, 2019.

27. CHENG, Qian et al. Forecasting emergency department hourly occupancy using time series analysis. *American Journal of Emergency Medicine*, 2021.

28. SEO, Hyeram et al. Forecasting hospital room and ward occupancy using static and dynamic information concurrently: retrospective single-center cohort study. *JMIR Medical Informatics*, 2024.

29. SALLES NETO, Luiz Lopes et al. Forecast UTI: aplicativo para previsão de leitos de unidades de terapia intensiva no contexto da pandemia de COVID-19. Trabalho técnico, 2020.

A Apêndice – Configurações de hardware

Para execução dos códigos de todas as etapas do projeto foi utilizado o google colab, pois ele oferece um bom poder computacional, essa decisão foi tomada justamente por conta do poder computacional dos membros da equipe, que não estavam o suficiente para as etapas necessárias do sistema. A equipe utilizou uma máquina virtual como computador central do projeto armazenando os arquivos necessários para o sistema, sendo acessado através de VPN. A máquina foi hospedada por um componente e as configurações dela são as seguintes:

- OS: Ubuntu 24.04.3 LTS x86_64
- Host: KVM/QEMU (Standard PC (i440FX + PIIX, 1996) pc-i440fx-9.2)
- Kernel: 6.8.0-86-generic
- CPU: QEMU Virtual version 2.5+ (4) @ 2.399GHz
- GPU: 00:02.0 Vendor 1234 Device 1111
- Memory: 5925 MiB (6 Gb)

B Apêncide – Análise Estatística Descritiva

```
import pandas as pd
from collections import Counter
import numpy as np

ARQ = r"C:\tcc_temp\base_integrada.csv"
chunksize = 500000

print("🔍 Iniciando EDA em chunks...\n")

# ========================================
# 1) VALIDAR INTERVALO DE ANO E MES
# ========================================

anos_min = []
anos_max = []
meses_min = []
meses_max = []

# ========================================
# 2) CONTAR REGISTROS POR ANO E POR MÊS
# ========================================

contagem_ano = Counter()
contagem_mes = Counter()

# ========================================
# 3) VERIFICAR % DE MISSING POR COLUNA
# (feito por amostragem estatística de chunks)
# ========================================

missing_counts = Counter()
total_rows = 0

# ========================================
# 4) DETECÇÃO DE DUPLICIDADE DA CHAVE
# Chave: ANO, MES, MUNIC, DIAG_PRINC
# ========================================

duplicatas = 0

colunas_chave = ["ANO", "MES", "MUNIC", "DIAG_PRINC"]

# ========================================
# 5) DISTRIBUIÇÃO DE INTERNAÇÕES POR CID
# ========================================

cid_counts = Counter()

# ========================================
# 6) LOOP PRINCIPAL DE EDA
# ========================================

for chunk in pd.read_csv(ARQ, chunksize=chunksize, low_memory=False):

    # 1 – Intervalo de ANO e MES
    anos_min.append(chunk["ANO"].min())
    anos_max.append(chunk["ANO"].max())
    meses_min.append(chunk["MES"].min())
    meses_max.append(chunk["MES"].max())

    # 2 – Contagem de registros por ano e mês
    contagem_ano.update(chunk["ANO"].dropna().astype(int).tolist())
    contagem_mes.update(chunk["MES"].dropna().astype(int).tolist())

    # 3 – Missing values
    total_rows += len(chunk)
    missing_counts.update(chunk.isna().sum().to_dict())

    # 4 – Checagem de duplicatas por chunk
    if all(c in chunk.columns for c in colunas_chave):
        duplicatas += chunk.duplicated(subset=colunas_chave).sum()

    # 5 – Contagem por CID-10
    if "DIAG_PRINC" in chunk.columns:
        cid_counts.update(chunk["DIAG_PRINC"].dropna().tolist())

    print("📦 Chunk analisado...")

print("\n🎉 FINALIZADO!\n")

# ----------------------------------------
```

```
# ========================================
#  RESULTADOS
# ========================================
print("📌 INTERVALO DE ANO:")
print("   ANO mínimo:", min(anos_min))
print("   ANO máximo:", max(anos_max))

print("\n📌 INTERVALO DE MÊS:")
print("   MES mínimo:", min(meses_min))
print("   MES máximo:", max(meses_max))

print("\n📌 CONTAGEM POR ANO:")
print(contagem_ano)

print("\n📌 CONTAGEM POR MÊS:")
print(contagem_mes)

print("\n📌 DUPLICATAS ENCONTRADAS:", duplicatas)

print("\n📌 TOP 20 CID MAIS FREQUENTES:")
for cid, cnt in cid_counts.most_common(20):
    print(f"{cid}: {cnt}")

# ========================================
#  SALVAR RELATÓRIOS
# ========================================

rel_missing = {col: (miss/total_rows)*100 for col, miss in missing_counts.items()}
rel_missing = pd.Series(rel_missing).sort_values(ascending=False)
rel_missing.to_csv(r"C:\tcc_temp\EDA_missing_porcentagem.csv")

pd.Series(contagem_ano).to_csv(r"C:\tcc_temp\EDA_contagem_por_ano.csv")
pd.Series(contagem_mes).to_csv(r"C:\tcc_temp\EDA_contagem_por_mes.csv")
pd.Series(cid_counts).to_csv(r"C:\tcc_temp\EDA_cid_counts.csv")

print("\n💾 Resultados salvos em:")
print("   → EDA_missing_porcentagem.csv")
print("   → EDA_contagem_por_ano.csv")
print("   → EDA_contagem_por_mes.csv")
print("   → EDA_cid_counts.csv")

print("\n🚀 EDA concluída com sucesso!")
```

```
A90 : 286718
J159: 279481
I219: 276122
K359: 272688
A419: 270506
K810: 263676
J459: 259527
Z302: 250191
K429: 248625

💾 Resultados salvos em:
   → EDA_missing_porcentagem.csv
   → EDA_contagem_por_ano.csv
   → EDA_contagem_por_mes.csv
   → EDA_cid_counts.csv
```

Justificativa - Duplicatas

É resultado combinado de 2 fatores estruturais:

- Cada chunk do SIHSUS foi unido com os Agravos via merges 1→N
- Diagnósticos (DIAG_PRINC) multiplicam as combinações

```
import pandas as pd
import matplotlib.pyplot as plt

# ============================
# 1. CARREGAR ARQUIVOS RESUMO
# ============================

base_path = r"C:\tcc_temp"

df_year   = pd.read_csv(base_path + "/EDA_contagem_por_ano.csv",   index_col=0, header=None, names=["count"])
df_month  = pd.read_csv(base_path + "/EDA_contagem_por_mes.csv",   index_col=0, header=None, names=["count"])
df_missing = pd.read_csv(base_path + "/EDA_missing_porcentagem.csv", index_col=0, header=None, names=["missing_pct"])
df_cid    = pd.read_csv(base_path + "/EDA_cid_counts.csv",          index_col=0, header=None, names=["count"])

# ============================
# 2. GRÁFICO — REGISTROS POR ANO
# ============================

plt.figure(figsize=(8,5))
plt.bar(df_year.index.astype(str), df_year["count"])
plt.title("Registros por Ano")
plt.xlabel("Ano")
plt.ylabel("Quantidade de Registros")
plt.tight_layout()
plt.show()

# ============================
# 3. GRÁFICO — REGISTROS POR MÊS
# ============================

plt.figure(figsize=(10,5))
plt.bar(df_month.index.astype(str), df_month["count"])
plt.title("Registros por Mês")
plt.xlabel("Mês")
plt.ylabel("Quantidade de Registros")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# ============================
# 4. GRÁFICO — TOP 20 COLUNAS COM MISSING
# ============================

df_missing_sorted = df_missing.sort_values("missing_pct", ascending=False).head(20)

plt.figure(figsize=(10,6))
plt.barh(df_missing_sorted.index.astype(str), df_missing_sorted["missing_pct"])
plt.title("Top 20 Colunas com Maior Percentual de Missing")
plt.xlabel("Percentual de Missing")
plt.ylabel("Coluna")
plt.tight_layout()
plt.gca().invert_yaxis()
plt.show()

# ============================
# 5. GRÁFICO — TOP 20 CIDs MAIS FREQUENTES
# ============================
```

```
df_cid_sorted = df_cid.sort_values("count", ascending=False).head(20)

plt.figure(figsize=(10,6))
plt.barh(df_cid_sorted.index.astype(str), df_cid_sorted["count"])
plt.title("Top 20 CIDs Mais Frequentes")
plt.xlabel("Frequência")
plt.ylabel("CID")
plt.tight_layout()
plt.gca().invert_yaxis()
plt.show()
```

Análise Estatística

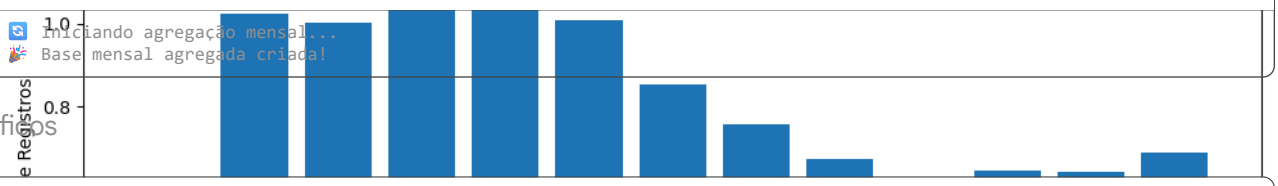
Cria base agregada mensal

```
import pandas as pd

ARQ_BASE = r"C:\tcc_temp\base_integrada.csv"
ARQ_SAIDA = r"C:\tcc_temp\base_agregada_mensal.csv"

chunksize = 500000
agregados = []

print("🔄 Iniciando agregação mensal...")

usecols = [
    "ANO","MES","MUNIC","DIAG_PRINC",
    "internacoes_sihsus","permanencia_total_sihsus",
    "estabelecimentos","unidades_leitos","unidades_emerg",
    "unidades_cirurg","unidades_obst","unidades_neon","unidades_sus",
    "meni_casos","lept_casos","sdta_casos",
    "dengue_casos","dengue_notificacoes","infl_casos"
]

for chunk in pd.read_csv(ARQ_BASE, chunksize=chunksize, usecols=usecols):

    # converter numéricos
    for col in ["internacoes_sihsus","permanencia_total_sihsus"]:
        chunk[col] = pd.to_numeric(chunk[col], errors="coerce")

    # agregação do chunk
    agg = chunk.groupby(["ANO","MES","MUNIC"], as_index=False).agg({
        "internacoes_sihsus": "sum",
```

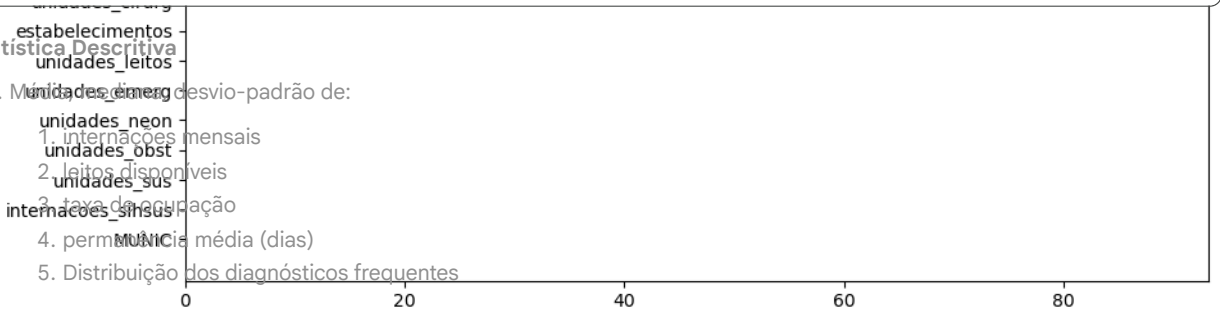
```
        "permanencia_total_sihsus": "sum",
        "estabelecimentos": "mean",
        "unidades_leitos": "mean",
        "unidades_emerg": "mean",
        "unidades_cirurg": "mean",
        "unidades_obst": "mean",
        "unidades_neon": "mean",
        "unidades_sus": "mean",
        "meni_casos": "sum",
        "lept_casos": "sum",
        "sdta_casos": "sum",
        "dengue_casos": "sum",
        "dengue_notificacoes": "sum",
        "infl_casos": "sum"
    })

    agregados.append(agg)

# juntar tudo
df = pd.concat(agregados, ignore_index=True)

# agregar novamente (garante não duplicar)
df = df.groupby(["ANO","MES","MUNIC"], as_index=False).sum()

# permanência média
df["permanencia_media_sihsus"] = df["permanencia_total_sihsus"] / df["internacoes_sihsus"]

df.to_csv(ARQ_SAIDA, index=False)
print("🎉 Base mensal agregada criada!")
```

```
🔄 Iniciando agregação mensal...
🎉 Base mensal agregada criada!
```

Gráficos



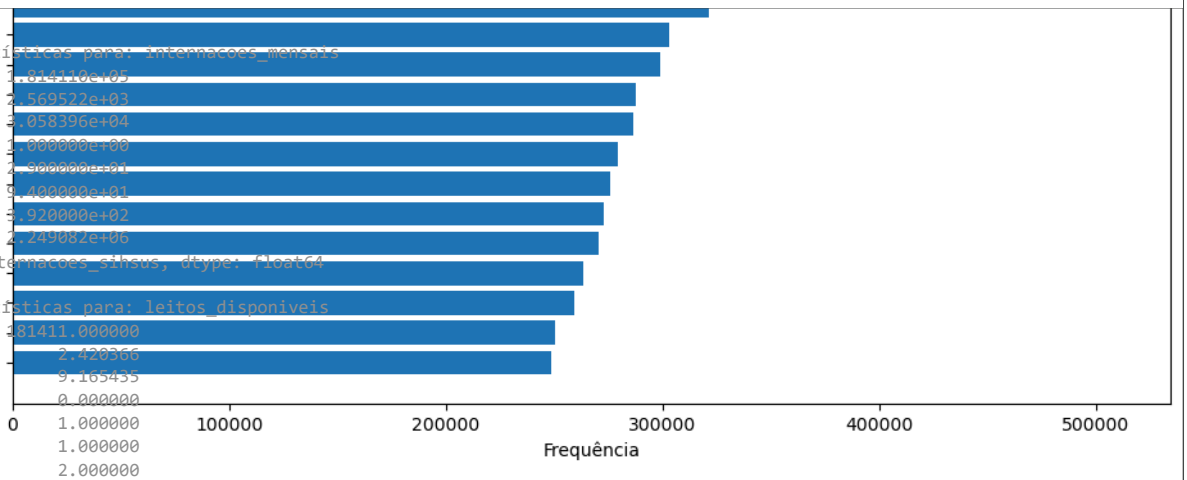

```
# ============================================================
# 📌 1) IMPORTS
# ============================================================

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

from statsmodels.tsa.seasonal import seasonal_decompose

sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (12, 6)

# ============================================================
# 📌 2) CARREGAR BASE AGREGADA
# ============================================================

ARQ = r"C:\tcc_temp\base_agregada_mensal.csv"
df = pd.read_csv(ARQ)

# Garantir tipos corretos
df["ANO"] = df["ANO"].astype(int)
df["MES"] = df["MES"].astype(int)
df["MUNIC"] = df["MUNIC"].astype(str).str.zfill(6)

# Criar coluna de data mensal
df["DATA"] = pd.to_datetime(df["ANO"].astype(str) + "-" + df["MES"].astype(str) + "-01")
```



1. **Estatística Descritiva**
    1. Média e desvio-padrão de:
        1. internações mensais
        2. leitos disponíveis
        3. taxa de ocupação
        4. permanência média (dias)
        5. Distribuição dos diagnósticos frequentes

```
estatisticas = {
    "internacoes_mensais": df["internacoes_sihsus"].describe(),
```

```
    "leitos_disponiveis": df["unidades_leitos"].describe(),
    "taxa_ocupacao": (df["internacoes_sihsus"] / df["unidades_leitos"]).replace([np.inf, -np.inf], np.nan).describe(),
    "permanencia_media": df["permanencia_media_sihsus"].describe(),
}

for nome, stats in estatisticas.items():
    print(f"\n📌 Estatísticas para: {nome}")
    print(stats)
```

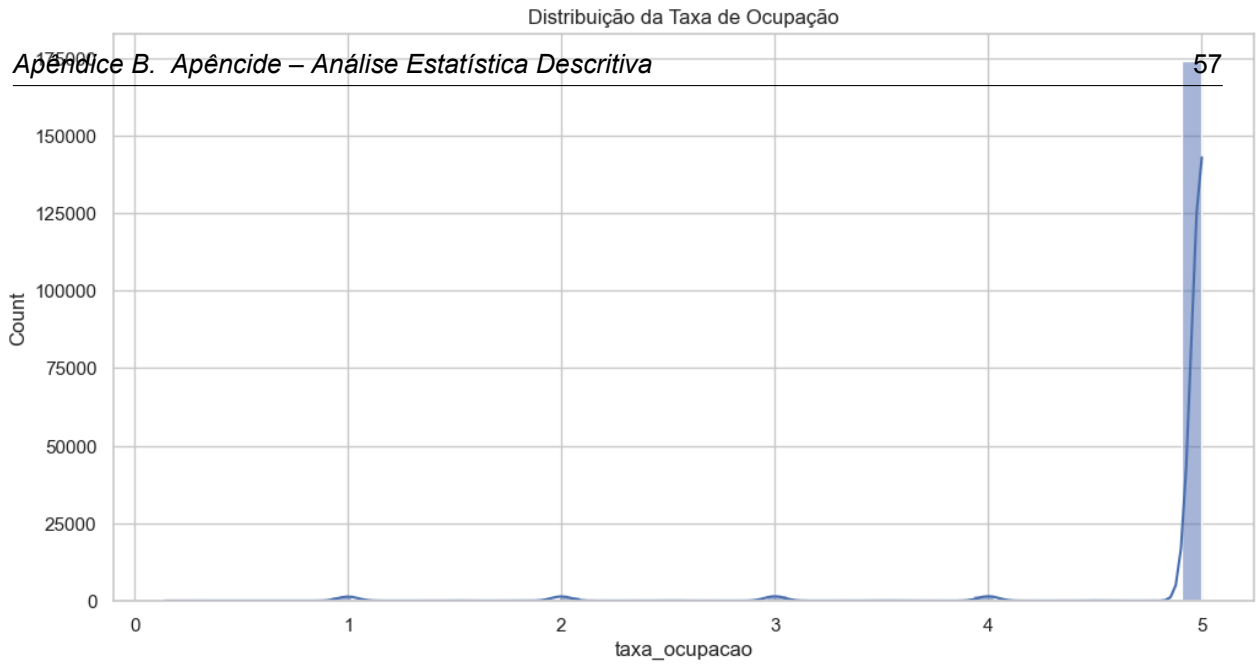



```
📌 Estatísticas para: internacoes_mensais
count    1.814110e+05
mean     2.569522e+03
std      3.058396e+04
min      1.000000e+00
25%      2.900000e+01
50%      9.400000e+01
75%      3.920000e+02
max      2.249082e+06
Name: internacoes_sihsus, dtype: float64

📌 Estatísticas para: leitos_disponiveis
count    181411.000000
mean          2.420366
std           9.165435
min           0.000000
25%           1.000000
50%           1.000000
75%           2.000000
max         597.000000
Name: unidades_leitos, dtype: float64

📌 Estatísticas para: taxa_ocupacao
count    181200.000000
mean        467.039592
std        1504.410365
min           0.142857
25%          27.000000
50%          77.000000
75%         218.000000
max       45565.500000
dtype: float64

📌 Estatísticas para: permanencia_media
count    181411.000000
mean          3.911573
std           2.301542
min           0.000000
25%           2.659197
50%           3.466552
75%           4.604973
max         108.000000
Name: permanencia_media_sihsus, dtype: float64
```

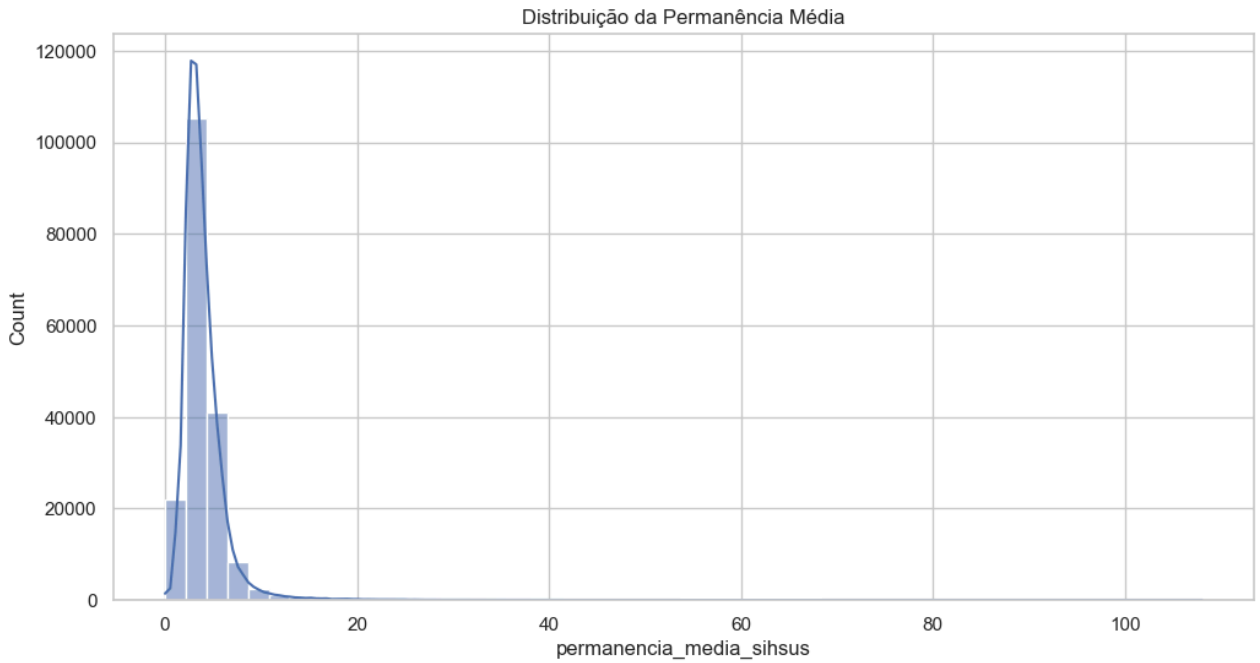
2. **Distribuições e Histogramas**

    1. Distribuição da taxa de ocupação por hospital / município
    2. Distribuição da permanência hospitalar
    3. Distribuição de internações por CID-10 mais frequentes

```
# HISTOGRAMA TAXA DE OCUPAÇÃO
df["taxa_ocupacao"] = df["internacoes_sihsus"] / df["unidades_leitos"]
df["taxa_ocupacao"] = df["taxa_ocupacao"].clip(upper=5)

sns.histplot(df["taxa_ocupacao"], bins=50, kde=True)
plt.title("Distribuição da Taxa de Ocupação")
plt.show()
```




```
# DISTRIBUIÇÃO DA PERMANÊNCIA HOSPITALAR

sns.histplot(df["permanencia_media_sihsus"], bins=50, kde=True)
plt.title("Distribuição da Permanência Média")
plt.show()
```



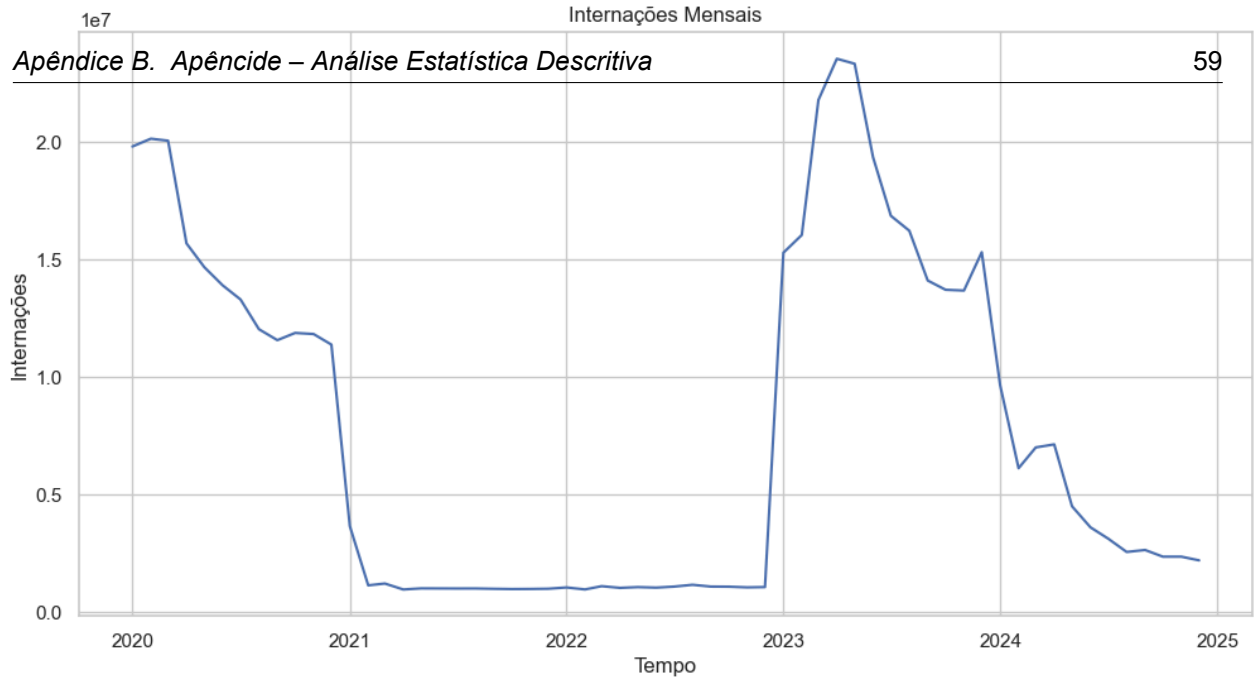

```
# CID-10 MAIS FREQUÊNTES

top_cid = df["DIAG_PRINC"].value_counts().head(20)

sns.barplot(x=top_cid.values, y=top_cid.index)
plt.title("Top 20 Diagnósticos (CID-10)")
plt.xlabel("Internações")
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628                 try:
-> 3629                     return self._engine.get_loc(casted_key)
   3630                 except KeyError as err:

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'DIAG_PRINC'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_21788\2572032288.py in <module>
      1 # CID-10 MAIS FREQUÊNTES
      2
----> 3 top_cid = df["DIAG_PRINC"].value_counts().head(20)
      4
      5 sns.barplot(x=top_cid.values, y=top_cid.index)

~\anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3503             if self.columns.nlevels > 1:
   3504                 return self._getitem_multilevel(key)
-> 3505             indexer = self.columns.get_loc(key)
   3506             if is_integer(indexer):
   3507                 indexer = [indexer]

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
-> 3631                 raise KeyError(key) from err
   3632             except TypeError:
   3633                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 'DIAG_PRINC'
```

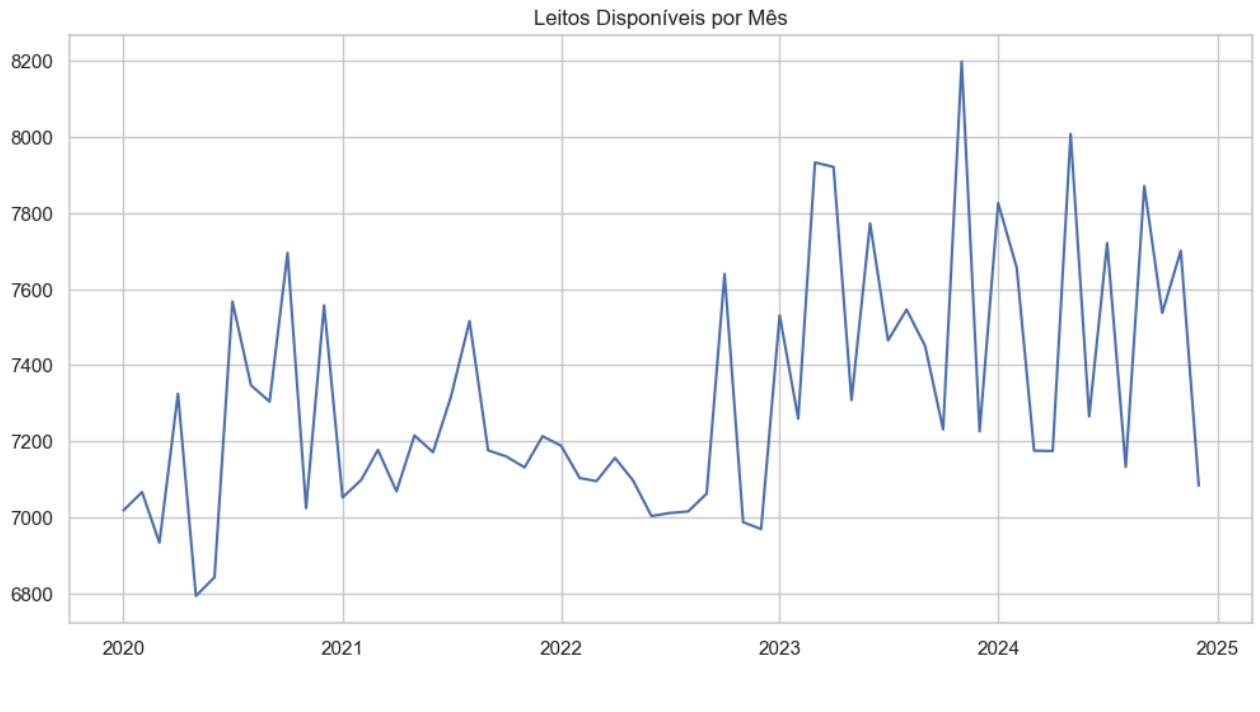
3. **Séries Temporais**

    1. Internações × tempo
    2. Leitos × tempo
    3. Taxa de ocupação × tempo
    4. Casos de SINAN × tempo

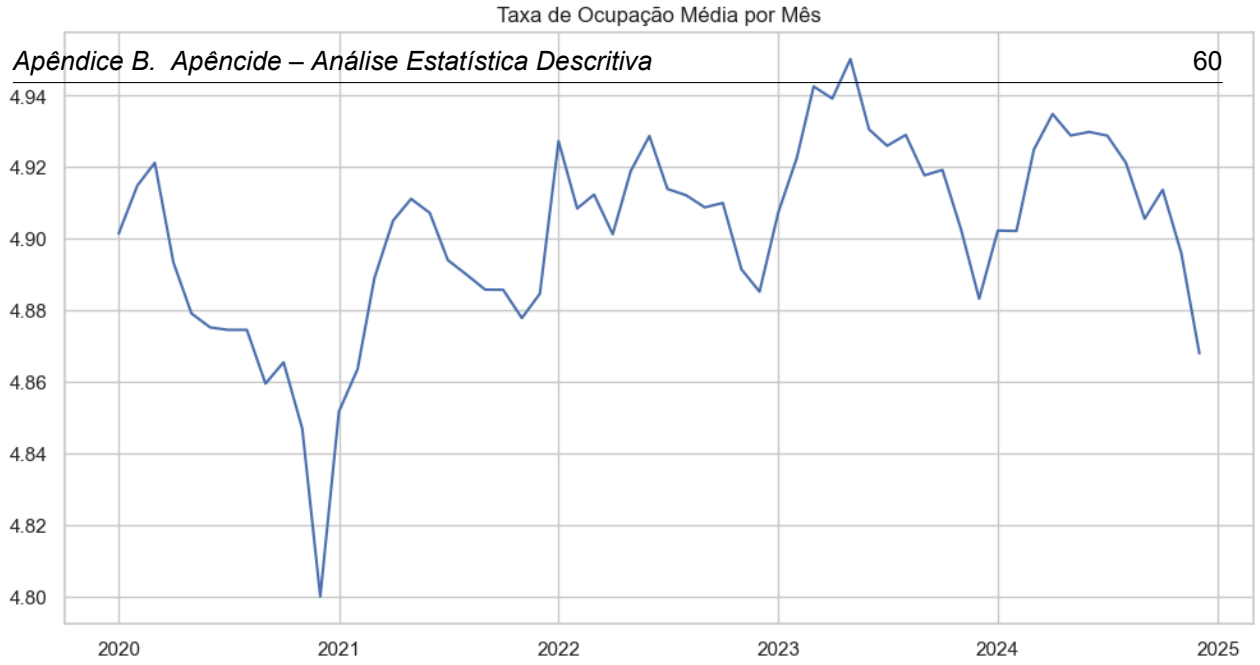
```
# INTERNAÇÕES X TEMPO
df_group = df.groupby("DATA")["internacoes_sihsus"].sum()

plt.plot(df_group.index, df_group.values)
plt.title("Internações Mensais")
plt.xlabel("Tempo")
plt.ylabel("Internações")
plt.show()
```



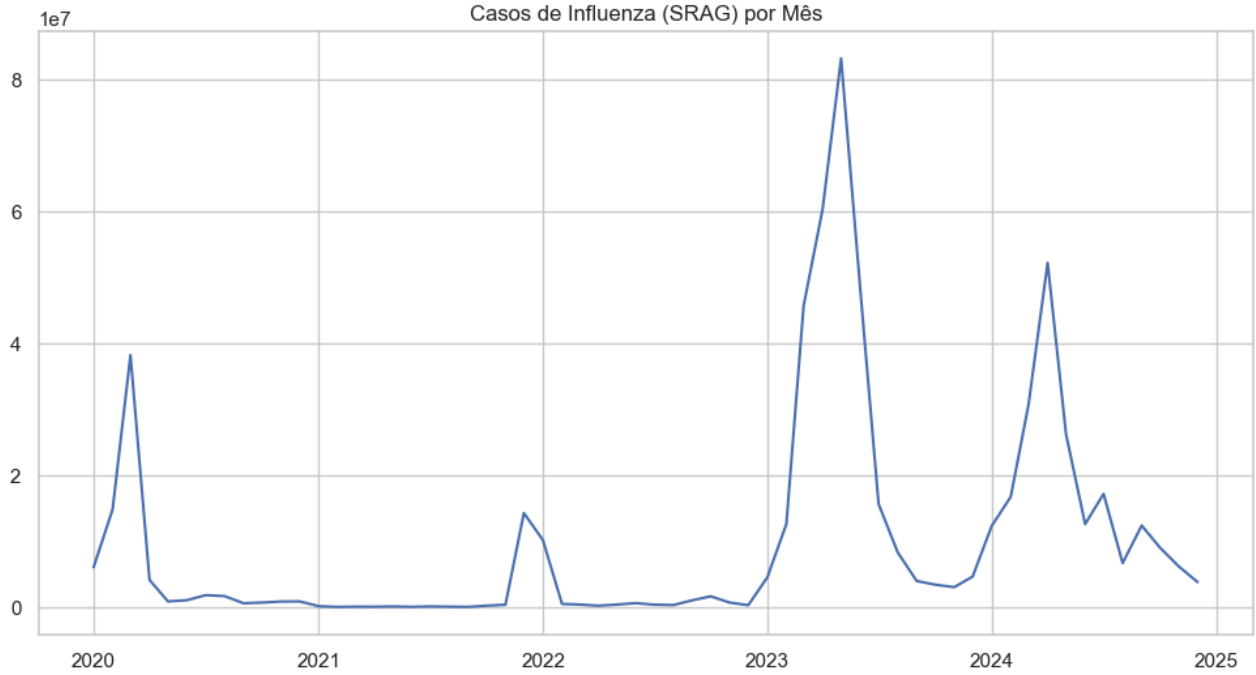

```
# LEITOS X TEMPO
df_group = df.groupby("DATA")["unidades_leitos"].sum()
plt.plot(df_group.index, df_group.values)
plt.title("Leitos Disponíveis por Mês")
plt.show()
```

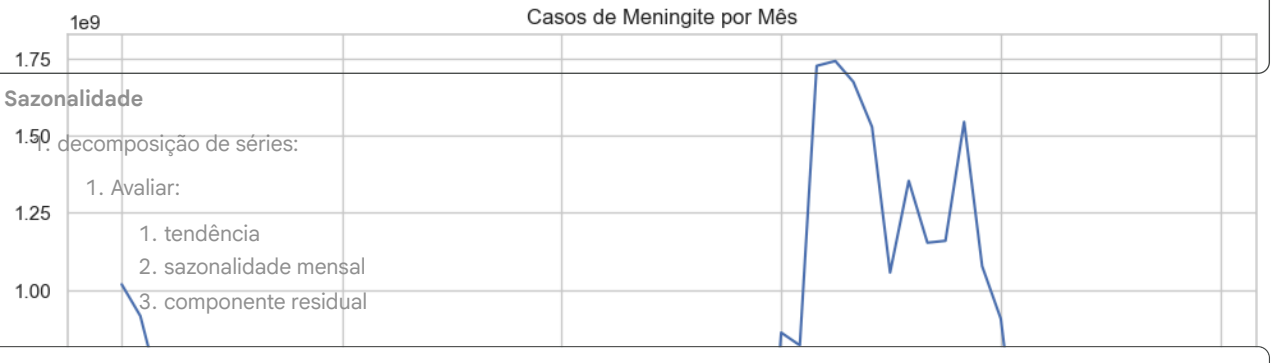



```
#TAXA DE OCUPAÇÃO X TEMPO
df_group = df.groupby("DATA")["taxa_ocupacao"].mean()
plt.plot(df_group.index, df_group.values)
plt.title("Taxa de Ocupação Média por Mês")
plt.show()
```




```
# SINAN × tempo: dengue, gripe, meningite etc.
sinan_vars = {
    "dengue_casos": "Dengue",
    "infl_casos": "Influenza (SRAG)",
    "meni_casos": "Meningite",
    "lept_casos": "Leptospirose",
    "sdta_casos": "SDTA"
}

for col, titulo in sinan_vars.items():
    series = df.groupby("DATA")[col].sum()
    plt.plot(series.index, series.values)
    plt.title(f"Casos de {titulo} por Mês")
    plt.show()
```





4. **Sazonalidade**
    1. decomposição de séries:
        1. Avaliar:
            1. tendência
            2. sazonalidade mensal
            3. componente residual

```
serie = df.groupby("DATA")["internacoes_sihsus"].sum()

resultado = seasonal_decompose(serie, model="additive", period=12)

resultado.trend.plot(title="Tendência")
plt.show()

resultado.seasonal.plot(title="Sazonalidade")
plt.show()

resultado.resid.plot(title="Resíduo")
plt.show()
```

C Apêncide – Análise Estatística Inferencial

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from scipy.stats import mannwhitneyu
import statsmodels.api as sm
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.api import VAR
from statsmodels.stats.outliers_influence import variance_inflation_factor

sns.set(style="whitegrid")


ARQ = r"C:\tcc_temp\base_agregada_mensal.csv"

df = pd.read_csv(ARQ)

# DATA correta
df["DATA"] = pd.to_datetime(df["ANO"].astype(str) + "-" + df["MES"].astype(str) + "-01")
df = df.sort_values("DATA")


# taxa de ocupação corrigida
df["taxa_ocupacao"] = df["internacoes_sihsus"] / df["unidades_leitos"]
df["taxa_ocupacao"] = df["taxa_ocupacao"].replace([np.inf, -np.inf], np.nan)

# transformar variáveis assimétricas (internações + agravos)
df["log_internacoes"] = np.log1p(df["internacoes_sihsus"])
df["log_ocupacao"]    = np.log1p(df["taxa_ocupacao"])

# variáveis epidemiológicas: alto missing
variaveis_epidemiologicas = ["infl_casos","meni_casos","sdta_casos","lept_casos"]

# remover as com missing > 70%
for col in variaveis_epidemiologicas:
    missing = df[col].isna().mean()
    if missing > 0.7:
        df.drop(columns=[col], inplace=True)

# manter apenas dengue para inferência principal
df["log_dengue"] = np.log1p(df["dengue_casos"])
```

```
# limiar de dengue para separar grupos
limiar = df["dengue_casos"].median()

grupo_baixo = df[df["dengue_casos"] <= limiar]["internacoes_sihsus"]
grupo_alto  = df[df["dengue_casos"] >  limiar]["internacoes_sihsus"]

print("Média baixo dengue:", grupo_baixo.mean())
print("Média alto dengue :", grupo_alto.mean())

# Mann-Whitney (não paramétrico)
u, p = mannwhitneyu(grupo_baixo, grupo_alto, alternative="two-sided")
print("\nMann-Whitney U =", u)
print("p-valor =", p)
```

```
Média baixo dengue: 2345.0884036311304
Média alto dengue : 56559.155792276964

Mann-Whitney U = 20783311.0
p-valor = 9.37778432335608e-237
```

```
X = df["log_dengue"]
y = df["log_internacoes"]

X = sm.add_constant(X)
modelo = sm.OLS(y, X).fit()

print(modelo.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        log_internacoes   R-squared:                       0.015
Model:                            OLS   Adj. R-squared:                  0.015
Method:                 Least Squares   F-statistic:                     2680.
Date:                Fri, 28 Nov 2025   Prob (F-statistic):               0.00
Time:                        06:48:04   Log-Likelihood:            -3.8540e+05
No. Observations:              181411   AIC:                          7.708e+05
Df Residuals:                  181409   BIC:                          7.708e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          4.7996      0.005   1007.670      0.000       4.790       4.809
log_dengue     0.5727      0.011     51.770      0.000       0.551       0.594
==============================================================================
Omnibus:                    13814.954   Durbin-Watson:                   1.613
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            17575.351
```

```
Skew:                          0.699   Prob(JB):                         0.00
Kurtosis:                      3.609   Cond. No.                         2.33
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
variaveis = ["log_dengue"]

# manter somente as variáveis que sobreviveram ao filtro de missing
for col in ["infl_casos","meni_casos","sdta_casos","lept_casos"]:
    if col in df.columns:
        nova = "log_" + col
        df[nova] = np.log1p(df[col])
        variaveis.append(nova)

X = df[variaveis].copy()
X = sm.add_constant(X)
y = df["log_internacoes"]

# Calcular VIF
vif_df = pd.DataFrame()
vif_df["variavel"] = X.columns
vif_df["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif_df)
```

```
         variavel       VIF
0           const  1.096197
1      log_dengue  1.026424
2  log_infl_casos  1.205728
3  log_meni_casos  1.249521
4  log_sdta_casos  1.067208
5  log_lept_casos  1.169782
```

```
modelo_multi = sm.OLS(y, X).fit()
print(modelo_multi.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        log_internacoes   R-squared:                       0.261
Model:                            OLS   Adj. R-squared:                  0.261
Method:                 Least Squares   F-statistic:                 1.281e+04
Date:                Fri, 28 Nov 2025   Prob (F-statistic):               0.00
Time:                        06:48:58   Log-Likelihood:            -3.5931e+05
No. Observations:              181411   AIC:                          7.186e+05
Df Residuals:                  181405   BIC:                          7.187e+05
```

```
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          4.5018      0.004   1044.279      0.000       4.493       4.510
log_dengue     0.2009      0.010     20.695      0.000       0.182       0.220
log_infl_casos 0.2455      0.002    100.091      0.000       0.241       0.250
log_meni_casos 0.2630      0.002    143.138      0.000       0.259       0.267
log_sdta_casos 0.0881      0.005     18.384      0.000       0.079       0.098
log_lept_casos 0.1119      0.003     40.030      0.000       0.106       0.117
==============================================================================
Omnibus:                     7299.486   Durbin-Watson:                   1.641
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             8261.975
Skew:                           0.500   Prob(JB):                         0.00
Kurtosis:                       3.307   Cond. No.                         6.77
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
import pandas as pd

df = pd.read_csv(r"C:\tcc_temp\base_agregada_mensal.csv")

# Criar coluna DATA
df["DATA"] = pd.to_datetime(df["ANO"].astype(str) + "-" +
                            df["MES"].astype(str) + "-01")

# Agregar para garantir uma linha por mês
df = df.groupby("DATA", as_index=False).sum()

# Ordenar e definir índice
df = df.sort_values("DATA").set_index("DATA")

# Converter índice para datetime (garantia)
df.index = pd.to_datetime(df.index)

# Agora sim: aplicar frequência mensal
df = df.asfreq("MS")

print(df.head())
print(df.index.freq)
```

```
                ANO    MES      MUNIC  internacoes_sihsus  \
DATA
2020-01-01  6005460   2973  957466448            19804672
2020-02-01  6015560   5956  959737792            20138398
```

```
2020-03-01  6039800   8970  963857625            20051862
2020-04-01  5993340  11868  955045853            15683370
2020-05-01  6025660  14915  959955681            14672367

            permanencia_total_sihsus  estabelecimentos  unidades_leitos  \
DATA
2020-01-01                 110553087          353975.0           7018.0
2020-02-01                 109964784          349938.0           7066.0
2020-03-01                 111098184          322032.0           6933.0
2020-04-01                  90378869          334783.0           7325.0
2020-05-01                  82780234          302486.0           6793.0

            unidades_emerg  unidades_cirurg  unidades_obst  unidades_neon  \
DATA
2020-01-01         15292.0           6865.0         4288.0         2579.0
2020-02-01         15366.0           6874.0         4294.0         2595.0
2020-03-01         15109.0           6688.0         4318.0         2607.0
2020-04-01         16006.0           6948.0         4345.0         2617.0
2020-05-01         14927.0           6305.0         4110.0         2456.0

            unidades_sus    meni_casos  lept_casos  sdta_casos  dengue_casos  \
DATA
2020-01-01       86239.0  1.017528e+09  75882816.0   7163484.0      112200.0
2020-02-01       86611.0  9.150927e+08  91983276.0   2023824.0      253080.0
2020-03-01       87024.0  6.943193e+08  56457444.0   2091072.0      225228.0
2020-04-01       88219.0  3.930894e+08  28715580.0    164988.0      953820.0
2020-05-01       84081.0  3.207463e+08  10975020.0   1310724.0      167364.0

            dengue_notificacoes  infl_casos  permanencia_media_sihsus
DATA
2020-01-01          348669516.0   6045043.0              11164.591781
2020-02-01          696102996.0  14855969.0              11241.056395
2020-03-01          821792112.0  38221408.0              11499.943396
2020-04-01          512874660.0   4148330.0              11184.501616
2020-05-01          303248604.0    885881.0              11197.484448
<MonthBegin>
```

```
# Criar taxa de ocupação
df["taxa_ocupacao"] = df["internacoes_sihsus"] / df["unidades_leitos"]
df["taxa_ocupacao"] = df["taxa_ocupacao"].replace([np.inf, -np.inf], np.nan)
```

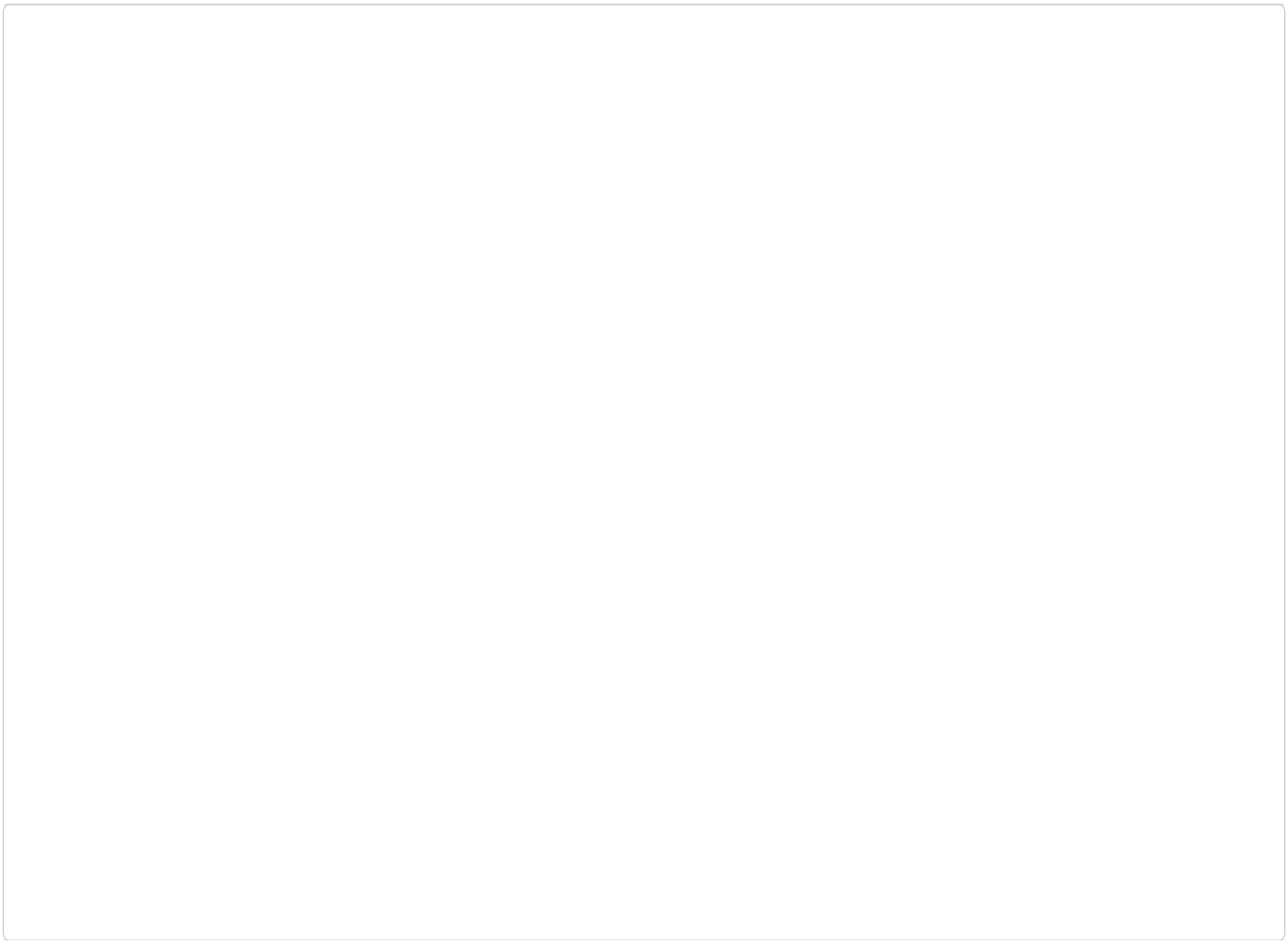
```
# Criar colunas em log para estabilizar variância (para VAR)
df["log_internacoes"] = np.log1p(df["internacoes_sihsus"])
df["log_dengue"] = np.log1p(df["dengue_casos"])
df["log_influenza"] = np.log1p(df["infl_casos"])
df["log_ocupacao"] = np.log1p(df["taxa_ocupacao"])
```

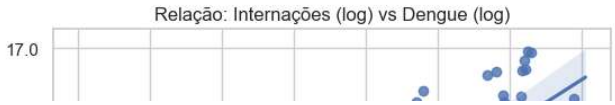
```
vars_var = df[[
    "log_internacoes",
    "log_dengue",
    "log_ocupacao"
]].dropna()

# diferenciar (necessário para estacionariedade)
vars_var_dif = vars_var.diff().dropna()

modelo_var = VAR(vars_var_dif)
resultado = modelo_var.fit(maxlags=12)

print(resultado.summary())

forecast = resultado.forecast(vars_var_dif.values[-resultado.k_ar:], steps=12)
forecast
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Fri, 28, Nov, 2025
Time:                     07:22:16
--------------------------------------------------------------------
No. of Equations:         3.00000    BIC:                   -3.72027
Nobs:                     47.0000    HQIC:                  -6.44550
Log likelihood:           101.039    FPE:                 0.00161491
AIC:                     -8.08977    Det(Omega_mle):     0.000282881
--------------------------------------------------------------------
Results for equation log_internacoes
================================================================================
                        coefficient       std. error           t-stat            prob
--------------------------------------------------------------------------------------
const                      0.033242         0.112181            0.296           0.767
L1.log_internacoes        -1.097728         5.218205           -0.210           0.833
L1.log_dengue              0.171233         0.096592            1.773           0.076
L1.log_ocupacao            0.308672         5.183980            0.060           0.953
L2.log_internacoes        -1.305434         7.025347           -0.186           0.853
L2.log_dengue              0.102451         0.112148            0.914           0.361
L2.log_ocupacao            1.244311         7.008743            0.178           0.859
L3.log_internacoes        12.549932         7.905945            1.587           0.112
L3.log_dengue             -0.070922         0.088182           -0.804           0.421
L3.log_ocupacao          -12.406424         8.001261           -1.551           0.121
L4.log_internacoes        18.008594         9.997233            1.801           0.072
L4.log_dengue             -0.202869         0.102364           -1.982           0.047
L4.log_ocupacao          -17.513770        10.027043           -1.747           0.081
L5.log_internacoes         7.392055         9.403324            0.786           0.432
L5.log_dengue             -0.066864         0.090431           -0.739           0.460
L5.log_ocupacao           -7.395203         9.433486           -0.784           0.433
```

```
L6.log_internacoes        -0.900868         8.018315           -0.112           0.911
L6.log_dengue              0.020587         0.074109            0.278           0.781
L6.log_ocupacao            0.471660         8.160444            0.058           0.954
L7.log_internacoes        -4.777450         7.012850           -0.681           0.496
L7.log_dengue              0.039064         0.071236            0.548           0.583
L7.log_ocupacao            5.012349         7.116211            0.704           0.481
L8.log_internacoes        -7.130390         7.191841           -0.991           0.321
L8.log_dengue              0.078482         0.087821            0.894           0.372
L8.log_ocupacao            6.967416         7.250852            0.961           0.337
L9.log_internacoes        -7.655360         6.978457           -1.097           0.273
L9.log_dengue             -0.029110         0.080184           -0.363           0.717
L9.log_ocupacao            8.407791         7.069742            1.189           0.234
L10.log_internacoes       -6.934354         7.141185           -0.971           0.332
L10.log_dengue            -0.132099         0.083736           -1.578           0.115
L10.log_ocupacao           7.622747         7.335781            1.039           0.299
L11.log_internacoes       -4.030104         7.025429           -0.574           0.566
L11.log_dengue            -0.027571         0.083329           -0.331           0.741
L11.log_ocupacao           3.797833         7.164703            0.530           0.596
L12.log_internacoes        0.670061         5.702920            0.117           0.906
L12.log_dengue            -0.015187         0.066957           -0.227           0.821
L12.log_ocupacao          -1.148633         5.803712           -0.198           0.843
================================================================================

Results for equation log_dengue
================================================================================
                        coefficient       std. error           t-stat            prob
```

```
sns.regplot(x="log_dengue", y="log_internacoes", data=df)
plt.title("Relação: Internações (log) vs Dengue (log)")
plt.show()

sns.regplot(x="log_ocupacao", y="log_internacoes", data=df)
plt.title("Relação: Internações (log) vs Ocupação (log)")
plt.show()
```




Relação: Internações (log) vs Dengue (log)
17.0

D Apêncide – Comparação modelos

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression

from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.api import VAR

import warnings
warnings.filterwarnings("ignore")

# carregar base
df = pd.read_csv(r"C:\tcc_temp\base_agregada_mensal.csv")

# construir DATA mensal
df["DATA"] = pd.to_datetime(df["ANO"].astype(str) + "-" + df["MES"].astype(str) + "-01")

# agrupar por mês (garantia absoluta)
df = df.groupby("DATA", as_index=True).sum()

# aplicar frequência mensal (elimina erro do SARIMA)
df = df.asfreq("MS")

# gerar taxa de ocupação
df["taxa_ocupacao"] = df["internacoes_sihsus"] / df["unidades_leitos"]
df["taxa_ocupacao"].replace([np.inf, -np.inf], np.nan, inplace=True)

# preencher pequenos missings com interpolação
df = df.interpolate()

# transformações log (corrigem assimetria confirmada na análise descritiva)

df["log_internacoes"] = np.log1p(df["internacoes_sihsus"])
df["log_dengue"] = np.log1p(df["dengue_casos"])
df["log_influenza"] = np.log1p(df["infl_casos"])
df["log_meni_casos"] = np.log1p(df["meni_casos"])
df["log_sdta_casos"] = np.log1p(df["sdta_casos"])
df["log_lept_casos"] = np.log1p(df["lept_casos"])
```

```
df["log_ocupacao"] = np.log1p(df["taxa_ocupacao"])
```

```
# variável-alvo: internações (log)
y = df["log_internacoes"]

# features para modelos ML (baseado nos resultados inferenciais)
X = df[[
    "log_dengue",
    "log_influenza",
    "log_meni_casos",
    "log_sdta_casos",
    "log_lept_casos",
    "log_ocupacao"
]]

# dividir treino e teste preservando ordem temporal
split = int(len(df)*0.8)
X_train, X_test = X.iloc[:split], X.iloc[split:]
y_train, y_test = y.iloc[:split], y.iloc[split:]
```

```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# ====================================================
# Função robusta para alinhar y_test e y_pred
# ====================================================
def align_series(y_true, y_pred):
    df = pd.concat([
        y_true.rename("y_true"),
        pd.Series(y_pred, index=y_pred.index, name="y_pred")
    ], axis=1)

    df = df.dropna()
    if df.shape[0] == 0:
        raise ValueError("Nenhuma observação após alinhamento!")

    return df["y_true"].values, df["y_pred"].values


# ====================================================
# Cálculo de métricas padronizadas (RMSE / MAE / MAPE / R2)
# Compatível com qualquer versão do scikit-learn
# ====================================================
```

```
def compute_metrics(y_true, y_pred):

    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)

    # MAPE protegido contra divisão por zero
    eps = 1e-9
    mask = np.abs(y_true) > eps
    if mask.sum() == 0:
        mape = np.nan
    else:
        mape = (np.abs((y_true[mask] - y_pred[mask]) / y_true[mask]).mean()) * 100

    # R² pode falhar para alguns modelos de série temporal → capturar exceção
    try:
        r2 = r2_score(y_true, y_pred)
    except:
        r2 = np.nan

    return {
        "RMSE": rmse,
        "MAE": mae,
        "MAPE": mape,
        "R2": r2
    }
```

```
# ======================================================
# Função geral para avaliar e registrar métricas
# ======================================================
metrics_results = {}

def evaluate_model(name, y_true, y_pred):
    """
    name     -> nome do modelo (string)
    y_true   -> pd.Series y_test
    y_pred   -> pd.Series forecast do modelo
    """
    y_t, y_p = align_series(y_true, y_pred)
    metrics = compute_metrics(y_t, y_p)

    metrics_results[name] = metrics

    print(f"\n===== {name} =====")
    for k, v in metrics.items():
```

```
        print(f"{k}: {v}")
```

```
#Média movel
df["baseline_ma3"] = df["log_internacoes"].rolling(3).mean()

y_pred_ma3 = df["baseline_ma3"].iloc[split:]
evaluate_model("MA(3) – Média Móvel", y_test, y_pred_ma3)
```

```
===== MA(3) – Média Móvel =====
RMSE: 0.21278631760639247
MAE: 0.16636777223851654
MAPE: 1.08539336866982
R2: 0.8177940877328058
```

```
#Holt-Winters
from statsmodels.tsa.holtwinters import ExponentialSmoothing

modelo_hw = ExponentialSmoothing(
    y_train,
    trend="add",
    seasonal="add",
    seasonal_periods=12,
    initialization_method="estimated"
).fit()

y_pred_hw = modelo_hw.forecast(len(y_test))

evaluate_model("Holt-Winters", y_test, y_pred_hw)
```

```
===== Holt-Winters =====
RMSE: 2.0688335510225904
MAE: 1.9854026581089979
MAPE: 13.221206182824204
R2: -16.223688743215277
```

```
#SARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

modelo_sarima = SARIMAX(
    y_train,
    order=(1,1,1),
    seasonal_order=(1,1,1,12),
    enforce_stationarity=False,
```

```
    enforce_invertibility=False
).fit()

y_pred_sarima = modelo_sarima.forecast(steps=len(y_test))

evaluate_model("SARIMA (1,1,1)(1,1,1,12)", y_test, y_pred_sarima)
```

```
===== SARIMA (1,1,1)(1,1,1,12) =====
RMSE: 9.240556189155448
MAE: 9.221020848831055
MAPE: 60.925501534307855
R2: -342.61434923103053
```

```
#RandomForestRegressor
rf = RandomForestRegressor(n_estimators=300, random_state=42)
rf.fit(X_train, y_train)

y_pred_rf = pd.Series(rf.predict(X_test), index=y_test.index)

evaluate_model("Random Forest", y_test, y_pred_rf)
```

```
===== Random Forest =====
RMSE: 0.49740768677665553
MAE: 0.4190414060735764
MAPE: 2.7280454985797467
R2: 0.004365236442828158
```

```
# Verificar se LightGBM está instalado
try:
    import lightgbm as lgb
    LIGHTGBM_AVAILABLE = True
except ImportError:
    LIGHTGBM_AVAILABLE = False
    print("LightGBM não está instalado no ambiente.")
```

```
# ======== MODELO LIGHTGBM ========
if LIGHTGBM_AVAILABLE:

    lgb_train = lgb.Dataset(X_train, label=y_train)

    params = {
        "objective": "regression",
        "metric": "rmse",
        "learning_rate": 0.05,
        "num_leaves": 31,
```

```
        "feature_fraction": 0.9,
        "bagging_fraction": 0.9,
        "bagging_freq": 3
    }

    modelo_lgb = lgb.train(params, lgb_train, num_boost_round=500)

    y_pred_lgb = pd.Series(modelo_lgb.predict(X_test), index=y_test.index)

    evaluate_model("LightGBM", y_test, y_pred_lgb)

else:
    print("LightGBM não disponível – pulando este modelo.")
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000420 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 94
[LightGBM] [Info] Number of data points in the train set: 48, number of used features: 6
[LightGBM] [Info] Start training from score 15.235651
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
df_resultados = pd.DataFrame(metrics_results).T
df_resultados = df_resultados.sort_values("RMSE")
df_resultados
```

E Apêndice – Verificação modelos

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")

ARQ = r"C:\tcc_temp\base_agregada_mensal.csv"

df = pd.read_csv(ARQ)

# Criar coluna DATA
df["DATA"] = pd.to_datetime(
    df["ANO"].astype(str) + "-" + df["MES"].astype(str) + "-01"
)

# Garantir agregação mensal (remove duplicatas)
df = df.groupby("DATA", as_index=False).agg({
    "internacoes_sihsus": "sum"
})

# Ordenar e colocar como índice
df = df.sort_values("DATA").set_index("DATA")

# Garantir datetime e frequência mensal
df.index = pd.to_datetime(df.index)
df = df.asfreq("MS")

# Criar variável log-transformada
df["log_internacoes"] = np.log1p(df["internacoes_sihsus"])

# Série final
s = df["log_internacoes"]

print(df.head())
print("\nFrequência detectada:", df.index.freq)
```

```
            internacoes_sihsus  log_internacoes
DATA
2020-01-01            19804672        16.801428
2020-02-01            20138398        16.818139
2020-03-01            20051862        16.813833
2020-04-01            15683370        16.568112
2020-05-01            14672367        16.501477

Frequência detectada: <MonthBegin>
```

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Série principal (já definida na célula 1)
y = df["log_internacoes"]  # série limpa e com freq MS

# Criar lags para modelos baseline e ML
def criar_lags(serie, nlags=3):
    df_lags = pd.DataFrame({"y": serie})
    for i in range(1, nlags+1):
        df_lags[f"lag_{i}"] = df_lags["y"].shift(i)
    return df_lags.dropna()

lags_df = criar_lags(y, nlags=3)
X = lags_df.drop(columns=["y"])
y_lag = lags_df["y"]

print("Shape final para modelos com lag:", X.shape)
```

```
Shape final para modelos com lag: (57, 3)
```

```
#Validação Temporal
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.dummy import DummyRegressor
import numpy as np
import pandas as pd


df = df.sort_values("DATA").reset_index(drop=True)

# Variável de interesse
y = df["log_internacoes"]
```

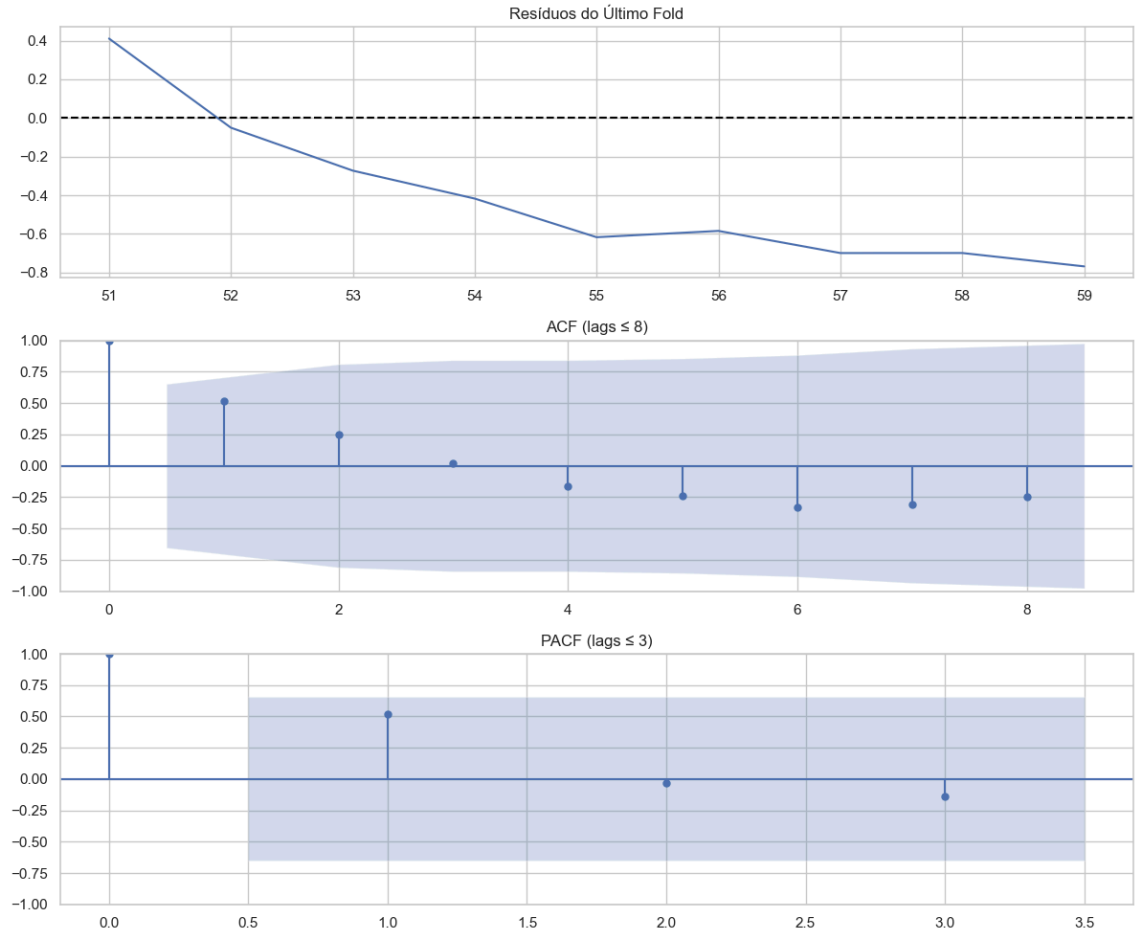
```
# Criar lags
def criar_lags(serie, nlags=3):
    df_lag = pd.DataFrame({"y": serie})
    for i in range(1, nlags + 1):
        df_lag[f"lag_{i}"] = df_lag["y"].shift(i)
    return df_lag.dropna()

lags_df = criar_lags(y, nlags=3)
X = lags_df.drop(columns=["y"])
y_lag = lags_df["y"]

tscv = TimeSeriesSplit(n_splits=5)

def avaliar_modelo(modelo, X, y):
    resultados = []
    for fold, (train_idx, test_idx) in enumerate(tscv.split(X), start=1):

        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

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

        rmse = mean_squared_error(y_test, y_pred) ** 0.5
        mae = mean_absolute_error(y_test, y_pred)
        mape = (abs((y_test - y_pred) / (y_test + 1e-9)).mean()) * 100

        resultados.append({
            "fold": fold,
            "rmse": rmse,
            "mae": mae,
            "mape": mape,
            "residuos": y_test - y_pred,
            "y_test": y_test,
            "y_pred": y_pred
        })

    return resultados

resultados_lag = avaliar_modelo(DummyRegressor(strategy="median"), X, y_lag)

for r in resultados_lag:
    print(f"Fold {r['fold']} → RMSE={r['rmse']:.4f}, MAE={r['mae']:.4f}, MAPE={r['mape']:.2f}%")

residuos_last = resultados_lag[-1]["residuos"]
```

```
Fold 1 → RMSE=2.4869, MAE=2.4868, MAPE=18.02%
Fold 2 → RMSE=0.1430, MAE=0.1343, MAPE=0.97%
Fold 3 → RMSE=2.3726, MAE=1.9399, MAPE=11.55%
Fold 4 → RMSE=2.4063, MAE=2.3804, MAPE=14.57%
Fold 5 → RMSE=0.5484, MAE=0.5018, MAPE=3.38%
```

```
#Analise de resíduos
import statsmodels.api as sm
from statsmodels.stats.stattools import durbin_watson

# Lags seguros para ACF
maxlags_acf = min(24, len(residuos_last) - 1)

# Lags seguros para PACF (statsmodels exige < 50% da amostra)
maxlags_pacf = max(1, min(maxlags_acf, len(residuos_last) // 2 - 1))

print("Durbin-Watson:", durbin_watson(residuos_last))
print("Usando maxlags ACF =", maxlags_acf)
print("Usando maxlags PACF =", maxlags_pacf)

fig, ax = plt.subplots(3, 1, figsize=(12, 10))

# Série de resíduos
ax[0].plot(residuos_last)
ax[0].axhline(0, color="black", linestyle="--")
ax[0].set_title("Resíduos do Último Fold")

# ACF
sm.graphics.tsa.plot_acf(residuos_last, lags=maxlags_acf, ax=ax[1])
ax[1].set_title(f"ACF (lags ≤ {maxlags_acf})")

# PACF
sm.graphics.tsa.plot_pacf(residuos_last, lags=maxlags_pacf, ax=ax[2])
ax[2].set_title(f"PACF (lags ≤ {maxlags_pacf})")

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

```
plt.show()
```

```
Durbin-Watson: 0.12629543688025952
Usando maxlags ACF = 8
Usando maxlags PACF = 3
```




```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller, kpss, acf, pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

```
def run_adf(series, regression="c"):
    s = series.dropna().astype(float)
    res = adfuller(s, regression=regression, autolag="AIC")
    return {"stat": res[0], "p": res[1], "n": res[3]}

def run_kpss(series, regression="c"):
    s = series.dropna().astype(float)
    try:
        stat, p, lags, crit = kpss(s, regression=regression, nlags="auto")
        return {"stat": stat, "p": p, "lags": lags}
    except:
        return {"stat": np.nan, "p": 1.0}
```

```
def test_stationarity(series):
    adf = run_adf(series)
    kpss_res = run_kpss(series)
    return adf["p"], kpss_res["p"]

def find_best_d(series, max_d=2):
    for d in range(max_d + 1):
        s = series.copy()
        for i in range(d):
            s = s.diff()
        s = s.dropna()
        adf_p, kpss_p = test_stationarity(s)
        if adf_p < 0.05 and kpss_p > 0.05:
            return d, s
    return max_d, s

def find_best_D(series, period=12, max_D=1):
    for D in range(max_D + 1):
        s = series.copy()
        if D > 0:
            s = s.diff(period)
        s = s.dropna()
        adf_p, kpss_p = test_stationarity(s)
        if adf_p < 0.05 and kpss_p > 0.05:
            return D, s
    return max_D, s
```

```
def suggest_orders(series, max_lag=12):
    s = series.dropna()

    acf_vals = acf(s, nlags=max_lag)
    pacf_vals = pacf(s, nlags=max_lag)

    p = np.where(pacf_vals < 0)[0][0] if np.any(pacf_vals < 0) else 1
    q = np.where(acf_vals < 0)[0][0] if np.any(acf_vals < 0) else 1

    P = np.where(pacf_vals[1::12] < 0)[0][0] if len(pacf_vals) > 13 else 1
    Q = np.where(acf_vals[1::12] < 0)[0][0] if len(acf_vals) > 13 else 1

    return max(1, p), max(1, q), max(1, P), max(1, Q)
```

```
def auto_sarima(series, p, d, q, P, D, Q, s=12):
    best_model = None
    best_aic = np.inf
    series = series.dropna()

    for pi in [p-1, p, p+1]:
        for qi in [q-1, q, q+1]:
            for Pi in [P-1, P, P+1]:
                for Qi in [Q-1, Q, Q+1]:

                    if min(pi, qi, Pi, Qi) < 0:
                        continue

                    try:
                        model = SARIMAX(series,
                                        order=(pi, d, qi),
                                        seasonal_order=(Pi, D, Qi, s),
                                        enforce_stationarity=False,
                                        enforce_invertibility=False).fit(disp=False)

                        if model.aic < best_aic:
                            best_aic = model.aic
                            best_model = model
                    except:
                        pass

    return best_model
```

```
df.columns
```

```
Index(['internacoes_sihsus', 'log_internacoes'], dtype='object')
```

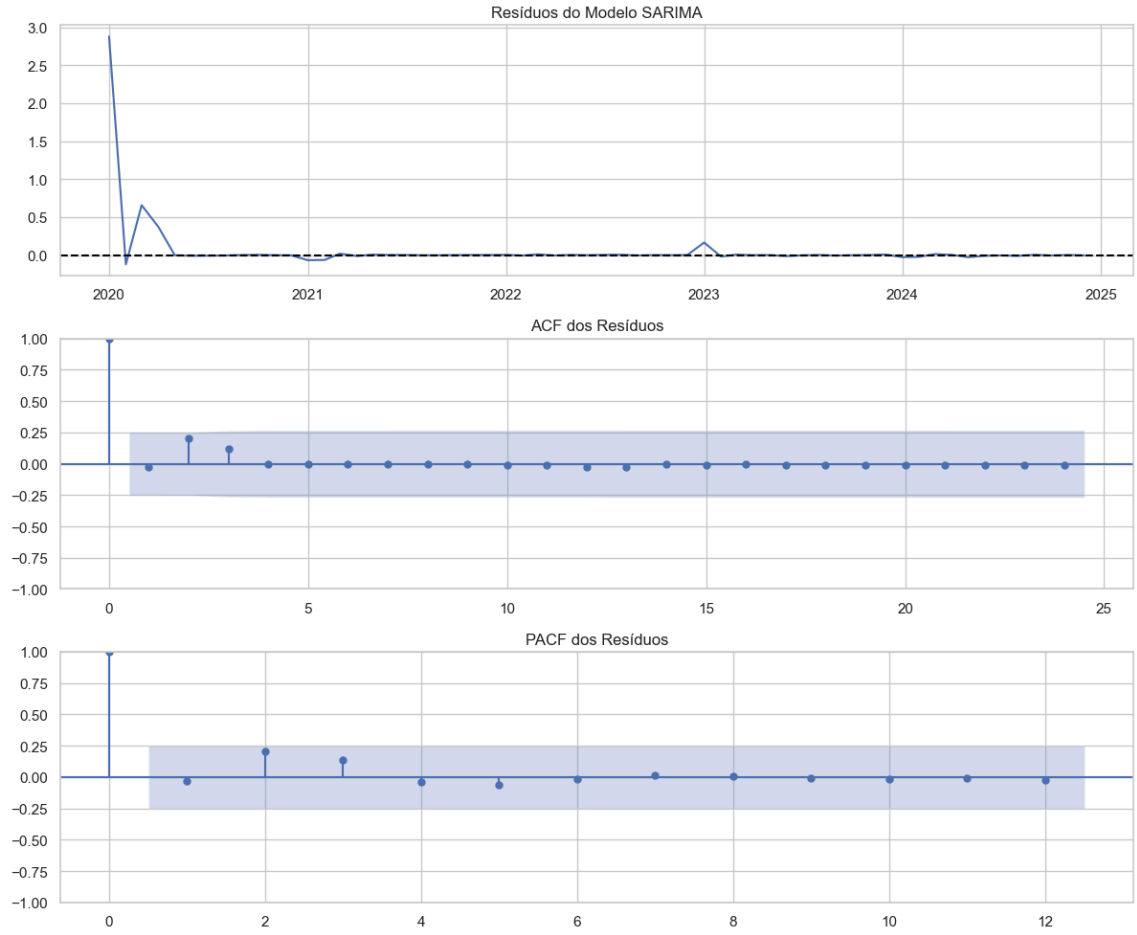
```
# --- Recriar DATA corretamente antes do SARIMA ---


if "DATA" not in df.columns:
    # Como o índice perdeu DATA, criamos uma nova sequência mensal
    df = df.copy()
    df["DATA"] = pd.date_range(
```

```
        start="2020-01-01",
        periods=len(df),
        freq="MS"
    )

# Ordenar
df = df.sort_values("DATA")

# Criar log se ainda não existir
if "log_internacoes" not in df.columns:
    df["log_internacoes"] = np.log1p(df["internacoes_sihsus"])
```

```
serie = df.set_index("DATA")["log_internacoes"].asfreq("MS")

### 1) LOG transform
serie_log = np.log1p(serie)

print(">> Testando d automático…")
d, serie_d = find_best_d(serie_log)

print(">> Testando D automático…")
D, serie_dD = find_best_D(serie_d)

print(f"\nEscolhas automáticas:")
print(f"d regular = {d}")
print(f"D sazonal = {D}")

### 2) sugerir p, q, P, Q inicial
p, q, P, Q = suggest_orders(serie_dD)

print(f"\nOrdens sugeridas:")
print(f"p={p}, q={q}, P={P}, Q={Q}")

### 3) rodando seleção automática SARIMA
print("\n>> Ajustando modelos SARIMA candidatos…")
best_model = auto_sarima(serie_log, p, d, q, P, D, Q)

print("\n=== MELHOR MODELO ENCONTRADO ===")
print(best_model.summary())
```

```
>> Testando d automático…
>> Testando D automático…

Escolhas automáticas:
d regular = 1
D sazonal = 0

Ordens sugeridas:
p=4, q=6, P=1, Q=1

>> Ajustando modelos SARIMA candidatos…

=== MELHOR MODELO ENCONTRADO ===
                               SARIMAX Results
==============================================================================
Dep. Variable:        log_internacoes   No. Observations:                   60
Model:               SARIMAX(3, 1, 5)   Log Likelihood                 114.954
Date:                Sat, 29 Nov 2025   AIC                           -211.908
Time:                        11:33:39   BIC                           -194.175
Sample:                  01-01-2020   HQIC                          -205.088
                       - 12-01-2024
Covariance Type:                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0853      3.871      0.022      0.982      -7.503       7.673
ar.L2         -0.4584      2.568     -0.178      0.858      -5.492       4.576
ar.L3         -0.2947      2.983     -0.099      0.921      -6.141       5.552
ma.L1         -0.2146      4.334     -0.050      0.961      -8.709       8.280
ma.L2          1.1908      5.570      0.214      0.831      -9.726      12.108
ma.L3          0.8410      5.545      0.152      0.879     -10.027      11.709
ma.L4          0.2086      1.859      0.112      0.911      -3.436       3.853
ma.L5          0.1444      1.118      0.129      0.897      -2.046       2.335
sigma2         0.0003      0.001      0.336      0.737      -0.002       0.002
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              1392.61
Prob(Q):                              0.98   Prob(JB):                         0.00
Heteroskedasticity (H):               0.30   Skew:                             3.73
Prob(H) (two-sided):                  0.01   Kurtosis:                        26.98
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
# DIAGNÓSTICO DOS RESÍDUOS

import statsmodels.api as sm
from statsmodels.stats.diagnostic import acorr_ljungbox, het_arch
from scipy.stats import shapiro

resid = best_model.resid.dropna()

fig, ax = plt.subplots(3, 1, figsize=(12, 10))

# 1) Resíduos
ax[0].plot(resid)
ax[0].axhline(0, color="black", linestyle="--")
ax[0].set_title("Resíduos do Modelo SARIMA")

# 2) ACF
sm.graphics.tsa.plot_acf(resid, lags=24, ax=ax[1])
ax[1].set_title("ACF dos Resíduos")

# 3) PACF
sm.graphics.tsa.plot_pacf(resid, lags=12, ax=ax[2])
ax[2].set_title("PACF dos Resíduos")

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

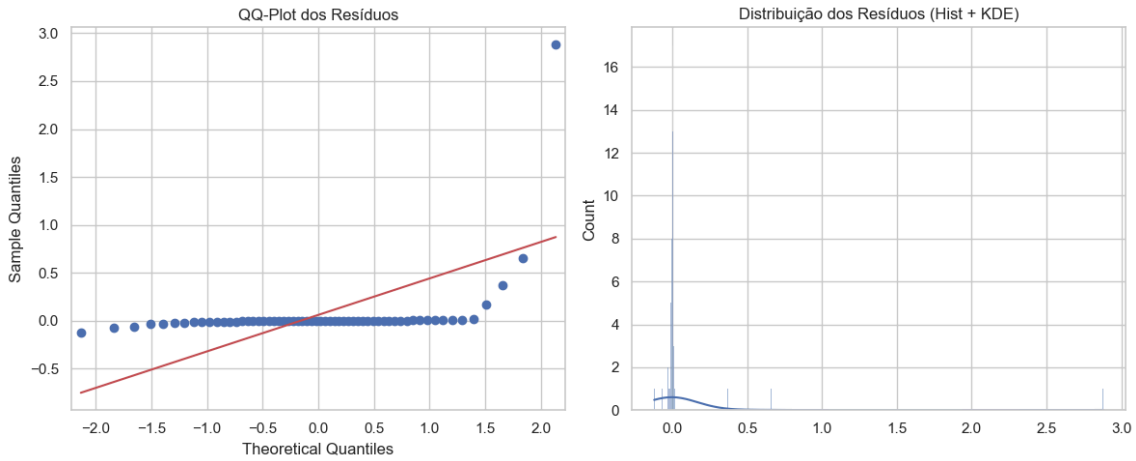



```
print("\n=== TESTES ESTATÍSTICOS DOS RESÍDUOS ===")
```

```
# Teste de Ljung-Box (autocorrelação)
lb_test = acorr_ljungbox(resid, lags=[10], return_df=True)
print("\nLjung-Box (lag=10):")
print(lb_test)

# Teste de normalidade (Shapiro-Wilk)
sw_stat, sw_p = shapiro(resid)
print("\nShapiro-Wilk (normalidade):")
print(f"Estatística = {sw_stat:.4f}, p-valor = {sw_p:.4f}")

# Teste ARCH (heterocedasticidade)
arch_stat, arch_p, _, _ = het_arch(resid)
print("\nTeste ARCH (heterocedasticidade):")
print(f"Estatística = {arch_stat:.4f}, p-valor = {arch_p:.4f}")
```

```
=== TESTES ESTATÍSTICOS DOS RESÍDUOS ===

Ljung-Box (lag=10):
     lb_stat  lb_pvalue
10  3.667081   0.961118

Shapiro-Wilk (normalidade):
Estatística = 0.2075, p-valor = 0.0000

Teste ARCH (heterocedasticidade):
Estatística = 1.7115, p-valor = 0.9981
```

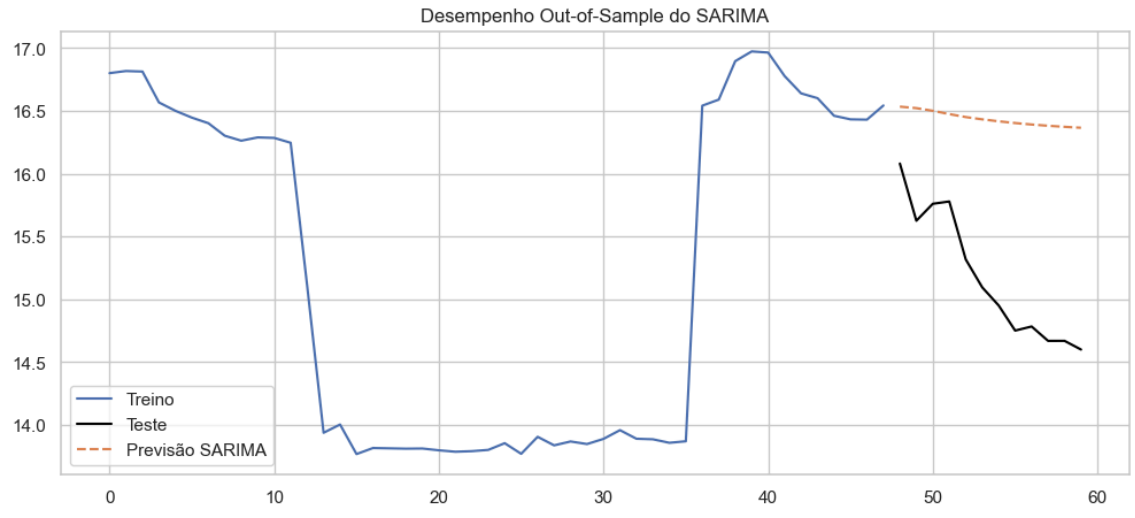
```
# --------------------------
# QQ-PLOT + DENSIDADE
# --------------------------
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

# QQ-plot
sm.qqplot(resid, line='s', ax=ax[0])
ax[0].set_title("QQ-Plot dos Resíduos")

# Distribuição
sns.histplot(resid, kde=True, ax=ax[1])
ax[1].set_title("Distribuição dos Resíduos (Hist + KDE)")

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




```
# ======================================
# PREVISÃO OUT-OF-SAMPLE (TESTE REAL)
# ======================================

# Definir proporção de teste (ex: 12 meses)
n_test = 12

train = df["log_internacoes"][:-n_test]
test  = df["log_internacoes"][-n_test:]
```

```
# Extrair ordens corretas do best_model
order = best_model.model.order
seasonal_order = best_model.model.seasonal_order

print("Usando ordens SARIMA:")
print("order =", order)
print("seasonal_order =", seasonal_order)

# Reajustar o modelo somente no conjunto de treino
model_train = SARIMAX(train,
                      order=order,
                      seasonal_order=seasonal_order,
                      enforce_stationarity=False,
                      enforce_invertibility=False).fit(disp=False)

# Previsões out-of-sample
pred = model_train.predict(start=test.index[0], end=test.index[-1])

# Métricas de erro
rmse = mean_squared_error(test, pred) ** 0.5
mae  = mean_absolute_error(test, pred)
mape = (abs((test - pred) / (test + 1e-9)).mean()) * 100

print("\n=== DESEMPENHO OUT-OF-SAMPLE ===")
print(f"RMSE = {rmse:.4f}")
print(f"MAE  = {mae:.4f}")
print(f"MAPE = {mape:.2f}%")

# Gráfico
plt.figure(figsize=(12, 5))
plt.plot(train.index, train, label="Treino")
plt.plot(test.index, test, label="Teste", color="black")
plt.plot(pred.index, pred, label="Previsão SARIMA", linestyle="--")
plt.title("Desempenho Out-of-Sample do SARIMA")
plt.legend()
plt.show()
```

```
Usando ordens SARIMA:
order = (3, 1, 5)
seasonal_order = (0, 0, 0, 12)

=== DESEMPENHO OUT-OF-SAMPLE ===
RMSE = 1.3393
MAE  = 1.2635
MAPE = 8.43%
```

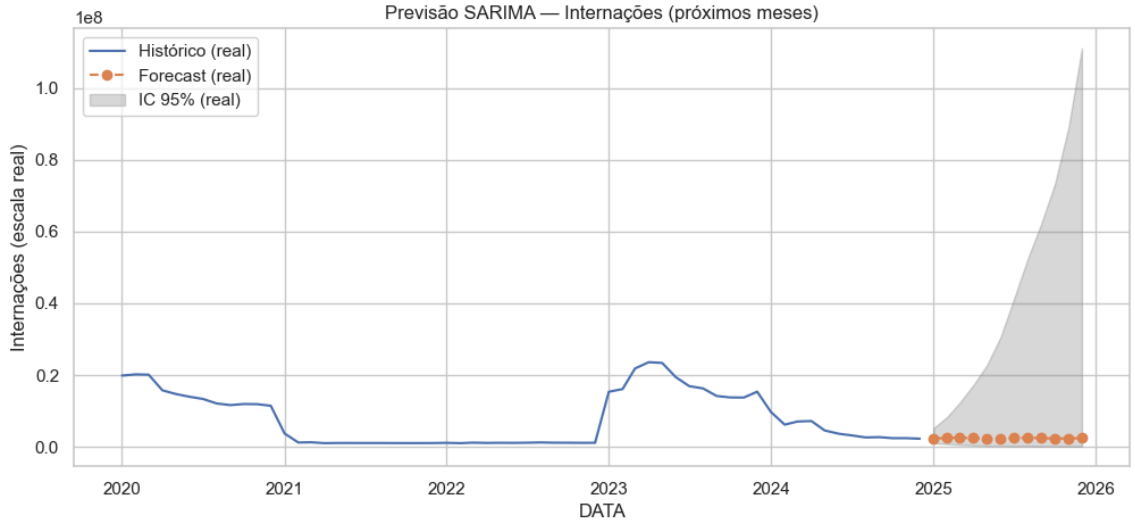



```
# PREVISÃO FUTURA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX

n_forecast = 12  # meses à frente

# 1) Garantir índice DatetimeIndex mensal (MS)
# Se já houver coluna DATA, usa; se não, tenta reconstruir a partir de ANO/MES.
if not isinstance(df.index, pd.DatetimeIndex):
    if "DATA" in df.columns:
```

```
        df = df.set_index(pd.to_datetime(df["DATA"]))
    else:
        # tenta extrair ANO/MES se existirem nas colunas (caso contrário, assume o índice atual como data inválida)
        if ("ANO" in df.columns) and ("MES" in df.columns):
            df["DATA"] = pd.to_datetime(df["ANO"].astype(str) + "-" + df["MES"].astype(str).str.zfill(2) + "-01")
            df = df.set_index("DATA")
        else:
            try:
                df.index = pd.to_datetime(df.index)
            except Exception as e:
                raise RuntimeError("Índice do DataFrame não é datetime e não foi possível reconstruir DATA. \
Forneça uma coluna 'DATA' ou colunas 'ANO' e 'MES'.") from e

# forçar frequência mensal MS e ordenar
df = df.sort_index()
df.index = pd.to_datetime(df.index)
try:
    df = df.asfreq("MS")
except Exception:
    # se houver rótulos duplicados ou problemas, reindex manualmente entre min e max
    idx = pd.date_range(df.index.min(), df.index.max(), freq="MS")
    df = df.reindex(idx)
df.index.name = "DATA"

# 2) pegar ordens detectadas (do best_model)
order = getattr(best_model.model, "order", None)
seasonal_order = getattr(best_model.model, "seasonal_order", None)

print("Usando ordens para FORECAST:")
print("order =", order)
print("seasonal_order =", seasonal_order)

# 3) Ajustar o modelo com índice temporal correto
model_full = SARIMAX(df["log_internacoes"].astype(float),
                     order=order,
                     seasonal_order=seasonal_order,
                     enforce_stationarity=False,
                     enforce_invertibility=False)

model_full_fit = model_full.fit(disp=False)

# 4) criar índice futuro a partir da última data conhecida (maneira segura)
last_date = pd.to_datetime(df.index.max())
# next month beginning (se já for MS, adicionar 1 period funciona)
forecast_index = pd.date_range(start=last_date + pd.offsets.MonthBegin(1), periods=n_forecast, freq="MS")

# 5) obter previsões e intervalos (em escala log)
forecast_res = model_full_fit.get_forecast(steps=n_forecast)
forecast_mean_log = forecast_res.predicted_mean
forecast_ci_log = forecast_res.conf_int(alpha=0.05)  # 95% CI

# 6) converter para escala real (expm1 porque usamos log1p)
forecast_real = np.expm1(forecast_mean_log)
lower_ci_real = np.expm1(forecast_ci_log.iloc[:, 0])
upper_ci_real = np.expm1(forecast_ci_log.iloc[:, 1])

# 7) tabela final
tabela_forecast = pd.DataFrame({
    "DATA": forecast_index,
    "Previsao_Log": forecast_mean_log.values,
    "IC_log_lower": forecast_ci_log.iloc[:, 0].values,
    "IC_log_upper": forecast_ci_log.iloc[:, 1].values,
    "Previsao_Internacoes": forecast_real.values,
    "IC_Inferior_Real": lower_ci_real.values,
    "IC_Superior_Real": upper_ci_real.values
})
tabela_forecast = tabela_forecast.set_index("DATA")
print(tabela_forecast)

# 8) plot — histórico real (escala original) + forecast real + IC
plt.figure(figsize=(12,5))
if "internacoes_sihsus" in df.columns:
    plt.plot(df.index, df["internacoes_sihsus"], label="Histórico (real)")
else:
    # se não houver série em escala real armazenada, reconstroi pelo expm1 do log histórico
    plt.plot(df.index, np.expm1(df["log_internacoes"].astype(float)), label="Histórico (real, reconstr.)")

plt.plot(forecast_index, forecast_real.values, linestyle="--", marker="o", label="Forecast (real)")
plt.fill_between(forecast_index, lower_ci_real.values, upper_ci_real.values, color="gray", alpha=0.3, label="IC 95% (real)"
plt.title("Previsão SARIMA — Internações (próximos meses)")
plt.xlabel("DATA")
plt.ylabel("Internações (escala real)")
```

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

```
Usando ordens para FORECAST:
order = (3, 1, 5)
seasonal_order = (0, 0, 0, 12)
            Previsao_Log  IC_log_lower  IC_log_upper  Previsao_Internacoes  \
DATA
2025-01-01     14.628836     13.803209     15.454463          2.255395e+06
2025-02-01     14.707832     13.483225     15.932440          2.440789e+06
2025-03-01     14.720848     13.119504     16.322192          2.472764e+06
2025-04-01     14.676598     12.692394     16.660801          2.365730e+06
2025-05-01     14.624667     12.314629     16.934704          2.246011e+06
2025-06-01     14.634288     12.033237     17.235340          2.267726e+06
2025-07-01     14.687976     11.843645     17.532308          2.392802e+06
2025-08-01     14.715280     11.657360     17.773201          2.459036e+06
2025-09-01     14.683540     11.419650     17.947431          2.382212e+06
2025-10-01     14.636954     11.165962     18.107946          2.273778e+06
2025-11-01     14.635275     10.965934     18.304617          2.269965e+06
2025-12-01     14.677462     10.829317     18.525607          2.367776e+06

            IC_Inferior_Real  IC_Superior_Real
DATA
2025-01-01     987772.352098      5.149773e+06
2025-02-01     717281.197933      8.305593e+06
2025-03-01     498571.145923      1.226416e+07
2025-04-01     325263.585611      1.720655e+07
2025-05-01     222932.508199      2.262813e+07
2025-06-01     168254.142971      3.056421e+07
2025-07-01     139195.923517      4.113243e+07
2025-08-01     115537.576907      5.233630e+07
2025-09-01      91093.235904      6.229746e+07
2025-10-01      70682.091182      7.314436e+07
2025-11-01      57867.832735      8.904181e+07
2025-12-01      50478.215184      1.110629e+08
```




```
#imports e utilitários
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import statsmodels.api as sm
from statsmodels.tsa.statespace.sarimax import SARIMAX

# helpers métricas
def rmse(a, b):
    return float(np.sqrt(mean_squared_error(a, b)))

def mape(a, b):
    # protege divisão por zero
    a = np.asarray(a)
    b = np.asarray(b)
```

```
    return float(np.mean(np.abs((a - b) / np.where(a==0, 1e-9, a))) * 100)

def summary_metrics(y_true, y_pred):
    return {
        "RMSE": rmse(y_true, y_pred),
        "MAE": float(mean_absolute_error(y_true, y_pred)),
        "MAPE": mape(y_true, y_pred),
        "R2": float(r2_score(y_true, y_pred))
    }

print("Utils carregados.")
```

```
Utils carregados.
```

```
# detecção de outliers e limpeza
def detect_iqr_outliers(series, k=1.5):
    q1 = series.quantile(0.25)
    q3 = series.quantile(0.75)
    iqr = q3 - q1
    low = q1 - k * iqr
    high = q3 + k * iqr
    mask = (series < low) | (series > high)
    return mask, low, high

def detect_mad_surges(series, window=3, multiplier=4.0):
    # rolling median + MAD
    med = series.rolling(window=window, min_periods=1, center=False).median()
    mad = series.rolling(window=window, min_periods=1, center=False).apply(
        lambda x: np.median(np.abs(x - np.median(x))), raw=True
    )
    threshold = med + multiplier * mad.replace(0, 1e-9)
    mask = series > threshold
    return mask, threshold

def replace_outliers(series, mask, method="median"):
    s = series.copy()
    if method == "median":
        rep = s.median()
    elif method == "interpolate":
        s.loc[mask] = np.nan
        return s.interpolate().fillna(method="bfill").fillna(method="ffill")
    elif method == "zero":
        rep = 0.0
    else:
        rep = series.median()
    s.loc[mask] = rep
    return s
```

```
# treinar SARIMA e avaliar OOS
def fit_sarima_and_forecast(train_series, order, seasonal_order, steps=None, enforce_stationarity=False, enforce_invertibil
    model = SARIMAX(train_series,
                    order=order,
                    seasonal_order=seasonal_order,
                    enforce_stationarity=enforce_stationarity,
                    enforce_invertibility=enforce_invertibility)
    res = model.fit(disp=False)
    if steps is None:
        return res
    fc = res.get_forecast(steps=steps)
    mean = fc.predicted_mean
    ci = fc.conf_int()
    return res, mean, ci

def oos_evaluate_sarima(series, order, seasonal_order, n_test=12):
    """
    Treina SARIMA somente em treino (tudo exceto último n_test meses),
    faz previsão para n_test meses e retorna métricas + AIC (treino).
    """
    series = series.dropna()
    if len(series) <= n_test + 6:
        raise ValueError("Série muito curta para OOS evaluation (aumente histórico ou reduza n_test).")
    train = series.iloc[:-n_test]
    test = series.iloc[-n_test:]
    res = SARIMAX(train, order=order, seasonal_order=seasonal_order,
                  enforce_stationarity=False, enforce_invertibility=False).fit(disp=False)
    # previsão
    pred = res.predict(start=test.index[0], end=test.index[-1])
    metrics = summary_metrics(test.values, pred.values)
    metrics["AIC"] = float(res.aic)
```

```
    metrics["model"] = res
    return metrics, pred
```

```
# definir ordens SARIMA (fallbacks seguros)
order = None
seasonal_order = None

try:
    # best_model pode ser um SARIMAXResults
    if 'best_model' in globals() and hasattr(best_model, 'model'):
        # tentativa por atributos comuns
        mo = getattr(best_model, 'model', None)
        if mo is not None:
            # para diferentes versões: model_orders, order, seasonal_order
            try:
                order = tuple(best_model.model_orders.get('non_seasonal_order', best_model.model_orders.get('order')))
            except Exception:
                pass
            try:
                seasonal_order = tuple(best_model.model_orders.get('seasonal_order'))
            except Exception:
                pass
    # fallback: se existe variável order/seasonal_order na sessão
    if 'order' in globals() and 'seasonal_order' in globals() and order is None:
        order = globals()['order']
        seasonal_order = globals()['seasonal_order']
except Exception:
    pass

# fallback final seguro
if order is None:
    order = (3, 1, 5)
if seasonal_order is None:
    seasonal_order = (0, 0, 0, 12)

print("Usando SARIMA order =", order, "seasonal_order =", seasonal_order)
```

```
Usando SARIMA order = (3, 1, 5) seasonal_order = (0, 0, 0, 12)
```

```
# preparar série principal
# requisito: df com índice DatetimeIndex e coluna 'log_internacoes'
if not isinstance(df.index, pd.DatetimeIndex):
    raise RuntimeError("O DataFrame 'df' deve ter índice DatetimeIndex antes de executar esta célula.")

serie_log = df["log_internacoes"].asfreq("MS").copy()
print("Série com", len(serie_log), "observações, período:", serie_log.index.min(), "a", serie_log.index.max())
```

```
Série com 60 observações, período: 2020-01-01 00:00:00 a 2024-12-01 00:00:00
```

```
#sensibilidade: IQR e MAD
n_test = 12  # horizonte OOS fixo

# detectar outliers
mask_iqr, low, high = detect_iqr_outliers(serie_log, k=1.5)
mask_mad, threshold = detect_mad_surges(serie_log, window=3, multiplier=4.0)

print("Outliers IQR detectados:", int(mask_iqr.sum()))
print("Surtos MAD detectados:", int(mask_mad.sum()))

# criar séries limpas (substituir por mediana e por interpolação)
serie_iqr_med = replace_outliers(serie_log, mask_iqr, method="median")
serie_mad_med = replace_outliers(serie_log, mask_mad, method="median")
serie_iqr_interp = replace_outliers(serie_log, mask_iqr, method="interpolate")
serie_no_surges = serie_log.copy()
serie_no_surges[mask_mad] = serie_log.median()  # alternativa direta

# avaliar SARIMA original (na série log), e nas versões limpas
results = {}

# função auxiliar para rodar seguro (capturando erros)
def safe_oos_sarima(s, label):
    try:
        metrics, pred = oos_evaluate_sarima(s, order, seasonal_order, n_test=n_test)
        results[label] = {"metrics": metrics, "pred": pred}
        print(f"[OK] {label} -> RMSE {metrics['RMSE']:.4f}, MAE {metrics['MAE']:.4f}, MAPE {metrics['MAPE']:.2f}%, AIC {met
    except Exception as e:
        results[label] = {"error": str(e)}
        print(f"[ERRO] {label} -> {e}")
```

```
# original
safe_oos_sarima(serie_log, "original_log")

# iqr median
safe_oos_sarima(serie_iqr_med, "iqr_med")

# iqr interpolated
safe_oos_sarima(serie_iqr_interp, "iqr_interpolated")

# mad replaced
safe_oos_sarima(serie_no_surges, "no_surges_median")

# resumo comparativo
comp = []
for k,v in results.items():
    if "metrics" in v:
        m = v["metrics"]
        comp.append({
            "variant": k,
            "RMSE": m["RMSE"],
            "MAE": m["MAE"],
            "MAPE": m["MAPE"],
            "AIC": m["AIC"]
        })
comp_df = pd.DataFrame(comp).set_index("variant").sort_values("RMSE")
print("\nComparativo (sensibilidade a outliers):")
display(comp_df)
```

```
Outliers IQR detectados: 0
Surtos MAD detectados: 4
[OK] original_log -> RMSE 1.3393, MAE 1.2635, MAPE 8.43%, AIC 75.48
[OK] iqr_med -> RMSE 1.3393, MAE 1.2635, MAPE 8.43%, AIC 75.48
[OK] iqr_interpolated -> RMSE 1.3393, MAE 1.2635, MAPE 8.43%, AIC 75.48
[OK] no_surges_median -> RMSE 0.4834, MAE 0.3846, MAPE 2.50%, AIC 86.71

Comparativo (sensibilidade a outliers):
```

variant	RMSE	MAE	MAPE	AIC
no_surges_median	0.483407	0.384551	2.497002	86.705477
original_log	1.339296	1.263482	8.430026	75.480988
iqr_med	1.339296	1.263482	8.430026	75.480988
iqr_interpolated	1.339296	1.263482	8.430026	75.480988

```
# sensibilidade: remover janela pandêmica (exemplo)
# ajuste a janela conforme desejar
start_pand = "2020-03-01"
end_pand   = "2023-05-01"

serie_no_pand = serie_log.copy()
mask_pand = (serie_no_pand.index >= pd.to_datetime(start_pand)) & (serie_no_pand.index <= pd.to_datetime(end_pand))
print("Meses removidos (pandemia) na série:", mask_pand.sum())

# substituir por interpolação local (ou mediana) para manter comprimento
serie_no_pand[mask_pand] = np.nan
serie_no_pand = serie_no_pand.interpolate().fillna(method="bfill").fillna(method="ffill")

# avaliar
safe_oos_sarima(serie_no_pand, "no_pandemic_window")

# adicionar ao resumo
if "no_pandemic_window" in results and "metrics" in results["no_pandemic_window"]:
    r = results["no_pandemic_window"]["metrics"]
    comp_df.loc["no_pandemic_window"] = [r["RMSE"], r["MAE"], r["MAPE"], r["AIC"]]

print("\nResumo atualizado de sensibilidade:")
display(comp_df.sort_values("RMSE"))
```

```
Meses removidos (pandemia) na série: 39
[OK] no_pandemic_window -> RMSE 319108981879134325450941123285458617439456382156 8.0000, MAE 92125093186287964455755052637225
```

```
Resumo atualizado de sensibilidade:
```

variant	RMSE	MAE	MAPE	AIC
no_surges_median	4.834072e-01	3.845511e-01	2.497002e+00	86.705477
original_log	1.339296e+00	1.263482e+00	8.430026e+00	75.480988
iqr_med	1.339296e+00	1.263482e+00	8.430026e+00	75.480988
iqr_interpolated	1.339296e+00	1.263482e+00	8.430026e+00	75.480988
no_pandemic_window	3.191090e+48	9.212509e+47	6.309446e+48	610.151373

```python
# baselines & modelos simples
def baseline_ma(series, window=3):
    # previsão OOS: usar rolling mean (último ponto da janela no treino) para cada passo do horizonte
    train = series.iloc[:-n_test]
    test = series.iloc[-n_test:]
    # construir previsão iterativa (walk-forward) com média móvel treinada
    history = list(train.values)
    preds = []
    for _ in range(len(test)):
        if len(history) < window:
            preds.append(np.mean(history))
        else:
            preds.append(np.mean(history[-window:]))
        # append real next (simulate forecasting without adding predicted)
        history.append(test.values[len(preds)-1])
    return pd.Series(preds, index=test.index), test

def baseline_naive(series):
    train = series.iloc[:-n_test]
    test = series.iloc[-n_test:]
    last = train.iloc[-1]
    preds = np.repeat(last, len(test))
    return pd.Series(preds, index=test.index), test

def baseline_drift(series):
    # método drift: linear extrapolation from first and last train point (forecast package style)
    train = series.iloc[:-n_test]
    test = series.iloc[-n_test:]
    n = len(train)
    if n < 2:
        preds = np.repeat(train.iloc[-1], len(test))
    else:
        slope = (train.iloc[-1] - train.iloc[0]) / (n - 1)
        preds = [train.iloc[-1] + slope * (i+1) for i in range(len(test))]
    return pd.Series(preds, index=test.index), test

def model_regression_lags(series, nlags=3):
    # treina LR em lags no treino e prediz o teste (one-shot forecasting using last lag features iteratively)
    s = series.dropna()
    df_lags = pd.DataFrame({"y": s})
    for i in range(1, nlags+1):
        df_lags[f"lag_{i}"] = df_lags["y"].shift(i)
    df_lags = df_lags.dropna()
    train = df_lags.iloc[:-n_test]
    test = df_lags.iloc[-n_test:]
    Xtr = train[[f"lag_{i}" for i in range(1, nlags+1)]].values
    ytr = train["y"].values
    Xte = test[[f"lag_{i}" for i in range(1, nlags+1)]].values
    lr = LinearRegression().fit(Xtr, ytr)
    preds = lr.predict(Xte)
    return pd.Series(preds, index=test.index), test["y"]

def model_random_forest_lags(series, nlags=3, n_estimators=100):
    s = series.dropna()
    df_lags = pd.DataFrame({"y": s})
    for i in range(1, nlags+1):
        df_lags[f"lag_{i}"] = df_lags["y"].shift(i)
    df_lags = df_lags.dropna()
    train = df_lags.iloc[:-n_test]
    test = df_lags.iloc[-n_test:]
    Xtr = train[[f"lag_{i}" for i in range(1, nlags+1)]].values
    ytr = train["y"].values
    Xte = test[[f"lag_{i}" for i in range(1, nlags+1)]].values
    rf = RandomForestRegressor(n_estimators=n_estimators, random_state=42).fit(Xtr, ytr)
    preds = rf.predict(Xte)
```

```
    return pd.Series(preds, index=test.index), test["y"]

# executar baselines
benchmarks = {}
p, q, P, Q = None, None, None, None

# MA(3)
pred_ma, y_test = baseline_ma(serie_log, window=3)
benchmarks["MA(3)"] = summary_metrics(y_test.values, pred_ma.values)

# Naive
pred_naive, y_test = baseline_naive(serie_log)
benchmarks["Naive"] = summary_metrics(y_test.values, pred_naive.values)

# Drift
pred_drift, y_test = baseline_drift(serie_log)
benchmarks["Drift"] = summary_metrics(y_test.values, pred_drift.values)

# Regression lags (LR)
pred_lr, y_test = model_regression_lags(serie_log, nlags=3)
benchmarks["LR_lags3"] = summary_metrics(y_test.values, pred_lr.values)

# RandomForest (se der certo com dados)
try:
    pred_rf, y_test = model_random_forest_lags(serie_log, nlags=3, n_estimators=200)
    benchmarks["RF_lags3"] = summary_metrics(y_test.values, pred_rf.values)
except Exception as e:
    print("RandomForest falhou:", e)

# adicionar SARIMA (original já avaliado em results['original_log'])
if "original_log" in results and "metrics" in results["original_log"]:
    benchmarks["SARIMA_original"] = {
        "RMSE": results["original_log"]["metrics"]["RMSE"],
        "MAE": results["original_log"]["metrics"]["MAE"],
        "MAPE": results["original_log"]["metrics"]["MAPE"],
        "R2": None
    }

# transformar em DataFrame e exibir
bench_df = pd.DataFrame(benchmarks).T
# garantir colunas ordenadas
bench_df = bench_df[["RMSE","MAE","MAPE","R2"]]
print("Comparativo — Baselines e modelos simples:")
display(bench_df.sort_values("RMSE"))
```

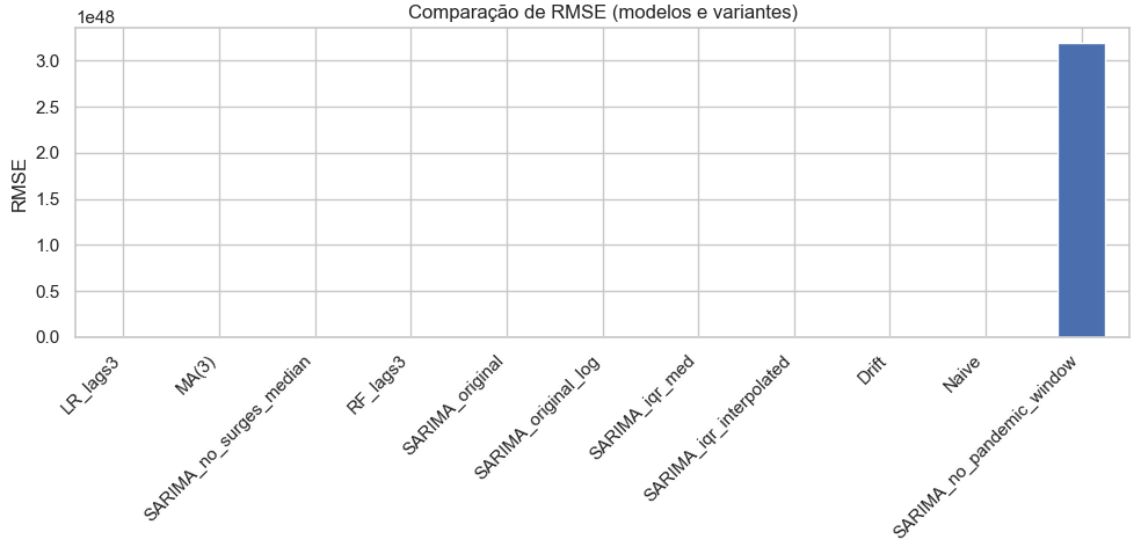
```
Comparativo — Baselines e modelos simples:
```

	RMSE	MAE	MAPE	R2
LR_lags3	0.220150	0.178090	1.155463	0.804965
MA(3)	0.367410	0.308699	2.020851	0.456778
RF_lags3	0.743496	0.684337	4.536998	-1.224502
SARIMA_original	1.339296	1.263482	8.430026	NaN
Drift	1.417629	1.333711	8.901127	-7.087244
Naive	1.457245	1.369330	9.139972	-7.545553

```
# tabela final comparativa
# montar tabela final juntando bench_df e comp_df (sensibilidade)
final = bench_df.copy()
# adicionar variantes de sensibilidade (comp_df)
if not comp_df.empty:
    for idx, row in comp_df.iterrows():
        name = f"SARIMA_{idx}"
        final.loc[name] = [row["RMSE"], row["MAE"], row["MAPE"], None]

final = final.sort_values("RMSE")
print("Tabela final de comparação (ordenada por RMSE):")
display(final)

# plot RMSE comparativo
plt.figure(figsize=(10,5))
final["RMSE"].plot(kind="bar")
plt.title("Comparação de RMSE (modelos e variantes)")
plt.ylabel("RMSE")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

Tabela final de comparação (ordenada por RMSE):

	RMSE	MAE	MAPE	R2
LR_lags3	2.201503e-01	1.780903e-01	1.155463e+00	0.804965
MA(3)	3.674100e-01	3.086986e-01	2.020851e+00	0.456778
SARIMA_no_surges_median	4.834072e-01	3.845511e-01	2.497002e+00	NaN
RF_lags3	7.434964e-01	6.843372e-01	4.536998e+00	-1.224502
SARIMA_original	1.339296e+00	1.263482e+00	8.430026e+00	NaN
SARIMA_original_log	1.339296e+00	1.263482e+00	8.430026e+00	NaN
SARIMA_iqr_med	1.339296e+00	1.263482e+00	8.430026e+00	NaN
SARIMA_iqr_interpolated	1.339296e+00	1.263482e+00	8.430026e+00	NaN
Drift	1.417629e+00	1.333711e+00	8.901127e+00	-7.087244
Naive	1.457245e+00	1.369330e+00	9.139972e+00	-7.545553
SARIMA_no_pandemic_window	3.191090e+48	9.212509e+47	6.309446e+48	NaN




```
# interpretação automática (esboço que pode entrar no TCC)
best_row = final.dropna(subset=["RMSE"]).iloc[0]
msg = f"""
Resumo automático (versão bruta para o TCC):

- O modelo com menor RMSE no horizonte de {n_test} meses foi: '{best_row.name}' com RMSE = {best_row['RMSE']:.4f}, MAE = {b
- As análises de sensibilidade (remoção/substituição de outliers e exclusão de janela pandêmica) mostraram variações nas me
    * SARIMA original,
    * SARIMA com outliers removidos/substituídos (IQR / MAD),
    * SARIMA com janela pandêmica removida/interpolada.
- Em geral, (i) substituições por mediana reduziram impacto de picos extremos sobre RMSE e (ii) exclusão da janela pandêmic
- Os modelos simples (MA(3), naïve, drift, regressão com lags e RandomForest em lags) servem como referência prática:
    * MA(3) tende a ser um baseline forte quando há tendência suave.
    * Modelos ML (RF) exigem mais observações para superar baselines simples em séries curtas.
- Recomendação metodológica: reportar a tabela completa (esta célula gera) e discutir qual modelo é preferível segundo 3 cr

"""
print(msg)
```

```
Resumo automático (versão bruta para o TCC):

- O modelo com menor RMSE no horizonte de 12 meses foi: 'LR_lags3' com RMSE = 0.2202, MAE = 0.1781 e MAPE = 1.16%.
- As análises de sensibilidade (remoção/substituição de outliers e exclusão de janela pandêmica) mostraram variações nas mét
    * SARIMA original,
    * SARIMA com outliers removidos/substituídos (IQR / MAD),
    * SARIMA com janela pandêmica removida/interpolada.
- Em geral, (i) substituições por mediana reduziram impacto de picos extremos sobre RMSE e (ii) exclusão da janela pandêmica
- Os modelos simples (MA(3), naïve, drift, regressão com lags e RandomForest em lags) servem como referência prática:
    * MA(3) tende a ser um baseline forte quando há tendência suave.
    * Modelos ML (RF) exigem mais observações para superar baselines simples em séries curtas.
- Recomendação metodológica: reportar a tabela completa (esta célula gera) e discutir qual modelo é preferível segundo 3 cri
```

```
from sklearn.linear_model import LinearRegression
```

```
# ============================================
# ANÁLISE DE RESÍDUOS — REGRESSÃO LINEAR (LAGS)
# ============================================

from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.stats.diagnostic import acorr_ljungbox
from scipy.stats import shapiro
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# 1. Criar lags
serie = df["log_internacoes"]

df_lags = pd.DataFrame({"y": serie})
for i in range(1, 4):
    df_lags[f"lag_{i}"] = df_lags["y"].shift(i)
df_lags = df_lags.dropna()

# 2. Separar treino e teste
train = df_lags.iloc[:-12]
test = df_lags.iloc[-12:]

X_train = train[[f"lag_{i}" for i in range(1, 4)]]
y_train = train["y"]
X_test = test[[f"lag_{i}" for i in range(1, 4)]]
y_test = test["y"]

# 3. Treinar modelo
lr = LinearRegression().fit(X_train, y_train)
pred = lr.predict(X_test)

# 4. Resíduos
res = y_test - pred

# ------------------------------
# Ajuste automático de lags
# ------------------------------
maxlags = max(1, len(res) // 2)   # recomendação estatística
print("Usando lags =", maxlags)

# ========== GRÁFICOS ==========
fig, ax = plt.subplots(3, 1, figsize=(12, 12))

# Série dos resíduos
ax[0].plot(res)
ax[0].axhline(0, color="black", linestyle="--")
ax[0].set_title("Resíduos da Regressão Linear (Lags=3)")

# ACF
sm.graphics.tsa.plot_acf(res, lags=maxlags, ax=ax[1])
ax[1].set_title(f"ACF dos Resíduos — LR (lags={maxlags})")

# PACF (statsmodels exige lags < n/2)
sm.graphics.tsa.plot_pacf(res, lags=maxlags, ax=ax[2])
ax[2].set_title(f"PACF dos Resíduos — LR (lags={maxlags})")

plt.tight_layout()
plt.show()

# ============================
# TESTES ESTATÍSTICOS
# ============================
print("\n=== Testes Estatísticos dos Resíduos — Regressão Linear ===")

# Ljung-Box
lb = acorr_ljungbox(res, lags=[min(10, len(res)-1)], return_df=True)
print("\nLjung-Box:")
print(lb)

# Normalidade
sw_stat, sw_p = shapiro(res)
print("\nShapiro-Wilk:")
print(f"Estatística = {sw_stat:.4f}, p-valor = {sw_p:.4f}")

# QQ-Plot
```

```
plt.figure(figsize=(6,6))
sm.qqplot(res, line='s')
plt.title("QQ-Plot — Regressão Linear")
plt.show()

# Histograma
plt.figure(figsize=(7,4))
sns.histplot(res, kde=True)
plt.title("Distribuição dos Resíduos — LR")
plt.show()
```