UNIVERSIDADE FEDERAL DE JUIZ DE FORA
FACULDADE DE CIÊNCIAS ECONÔMICAS

Hugo Monteiro de Pinho

Investigação da capacidade preditiva de Redes Neurais Recorrentes "Long Short-Term Memory" (LSTM) para os preços de fechamento do Mini Índice de 2022 a 2024

JUIZ DE FORA
2025

Hugo Monteiro de Pinho

Avaliando a capacidade preditiva de Redes Neurais Recorrentes “Long Short-Term Memory” (LSTM) para os preços de fechamento do Mini Índice de 2022 a 2024

Monografia apresentada ao curso de Ciências Econômicas da Universidade Federal de Juiz de Fora, como requisito parcial à obtenção do título de bacharel em Ciências Econômicas.

Orientador: Prof. Dr. Paulo César Coimbra

JUIZ DE FORA

2025

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

de Pinho, Hugo Monteiro.
Avaliando a capacidade preditiva de Redes Neurais Recorrentes "Long Short-Term Memory" (LSTM) para os preços de fechamento do Mini Índice de 2022 a 2024 / Hugo Monteiro de Pinho. -- 2025.
78 p.

Orientador: Paulo César Coimbra
Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Juiz de Fora, Faculdade de Economia, 2025.

1. Previsão. 2. Rede Neural. 3. Bovespa. 4. Mercado Financeiro. I. Coimbra, Paulo César , orient. II. Título.

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
REITORIA - FACECON - Depto. de Economia

FACULDADE DE ECONOMIA / UFJF

ATA DE APROVAÇÃO DE MONOGRAFIA II

NA DATA DE 11/03/2025, A BANCA EXAMINADORA, COMPOSTA PELOS PROFESSORES:

1 – PAULO CÉSAR COIMBRA LISBÔA - ORIENTADOR; E

2 – ROGÉRIO SILVA DE MATTOS,

REUNIU-SE PARA AVALIAR A MONOGRAFIA DO ACADÊMICO HUGO MONTEIRO DE PINHO, INTITULADA:

LONG SHORT-TERM MEMORY" (LSTM) PARA OS PREÇOS DO MINI ÍNDICE DE 2022 A 2024.

APÓS PRIMEIRA AVALIAÇÃO, RESOLVEU A BANCA SUGERIR ALTERAÇÕES AO TEXTO APRESENTADO, CONFORME RELATÓRIO SINTETIZADO PELO ORIENTADOR. A BANCA, DELEGANDO AO ORIENTADOR A OBSERVÂNCIA DAS ALTERAÇÕES PROPOSTAS, RESOLVEU APROVAR A REFERIDA MONOGRAFIA.

Documento assinado eletronicamente por **Paulo César Coimbra Lisbôa**, **Professor(a)**, em 17/03/2025, às 16:47, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Rogerio Silva de Mattos**, **Professor(a)**, em 17/03/2025, às 18:51, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **2296865** e o código CRC **CA66F388**.

RESUMO

Uma das estratégias utilizadas por operadores e investidores do mercado financeiro para maximizar os rendimentos de suas carteiras é a previsão dos preços futuros dos ativos, seja com base na análise histórica dos preços ou nos fundamentos das empresas. Neste estudo, avaliamos a capacidade preditiva de uma rede neural do tipo LSTM na previsão dos preços de fechamento do Mini Índice Bovespa, testando quatro modelos distintos. O primeiro, mais simples, considera apenas o preço de fechamento como variável. O segundo incorpora mais informações de preço. O terceiro inclui as informações de preço e um único indicador técnico. Por fim, o quarto e mais complexo modelo combina dados de preço com múltiplos indicadores técnicos. A comparação entre os modelos permite avaliar a eficácia do uso de diferentes conjuntos de variáveis na previsão do comportamento do mercado.

Palavras-chave: Previsão, Rede Neural, Bovespa, Mercado Financeiro

ABSTRACT

One of the strategies used by financial market traders and investors to maximize portfolio returns is forecasting future asset prices, either based on historical price analysis or company fundamentals. In this study, we assess the predictive capability of an LSTM neural network in forecasting the closing prices of the Mini Bovespa Index, testing four different models. The first and simplest model considers only the closing price as a variable. The second incorporates additional price-related information. The third includes price data along with a single technical indicator. Finally, the fourth and most complex model combines price data with multiple technical indicators. Comparing these models allows us to evaluate the effectiveness of different variable sets in predicting market behavior.

Keywords: Forecasting, Neural Network, Bovespa, Financial Market

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

Adam - *Adaptive Moment Estimation*
ANN - *Artificial Neural Networks*
EMA - *Exponential Moving Average*
LLM - *Large Language Model*
LSTM - *Long-Short Term Memory*
MACD - *Moving Average Convergence Divergence*
MAE - *Mean Absolute Error*
MAPE - *Mean Average Percentage Error*
MSELoss - *Mean Squared Error Loss*
OBV - *On-Balance Volume*
ReLU - *Rectified Linear Unit*
LeakyReLU – Leaky Rectified Linear Unit
RMSprop - *Root Mean Square Propagation*
RNN - *Recurrent Neural Network*
RSI - *Relative Strength Index*
SMA - *Simples Moving Average*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, houve uma explosão no campo da inteligência artificial e do aprendizado de máquina, tanto em termos de aplicações que começam a infiltrar nosso cotidiano quanto em avanços científicos significativos (LUDERMIR, 2021). E, com essa proliferação cada vez maior, todos os setores da economia sentem a aproximação de uma mudança na forma como negócios são feitos, e buscam implementar esses avanços no seu modelo de negócio e operação com propósito de aumentar eficiência, aumentar a satisfação dos clientes e, ultimamente, aumentar os lucros.

Naturalmente, um desses setores que não poderia ficar de fora é o Mercado Financeiro que, segundo pesquisa conduzida pelo World Economic Forum, 85% dos participantes já implementaram inteligência artificial de alguma forma (WORLD ECONOMIC FORUM; CAMBRIDGE CENTRE FOR ALTERNATIVE FINANCE, 2020) em suas operações e podem ser aplicadas ao gerenciamento de risco, compliance e trading COMMODITY FUTURES TRADING COMMISSION, 2019). O uso da computação no mercado financeiro já possui uma longa trajetória e está amplamente consolidado.

Nos anos 1990 e 2000, com o avanço da capacidade computacional e o aumento da disponibilidade de dados, os primeiros métodos de aprendizado de máquina começaram a ser aplicados no setor. Estratégias como o High-Frequency Trading (HFT) só se tornaram viáveis devido a esses avanços tecnológicos, permitindo a execução de ordens em altíssima velocidade (Hawkins, 2012).

À medida que a computação evoluía, novas técnicas eram continuamente incorporadas ao mercado financeiro, ampliando a automação, a análise preditiva e a eficiência das operações. E, conforme progresso era feito na computação, cada vez mais métodos eram incorporados ao setor financeiro. E, uma das metodologias que já foi incorporada pelo mercado, são as previsões com redes neurais (ZHANG, 2003). Assim, este estudo busca investigar se a adição de indicadores técnicos consegue superar um modelo que utiliza apenas o valor de fechamento para realizar suas previsões.

Tais indicadores fazem uso de dados como preço de fechamento, abertura, máxima, mínima e o volume de negociação para auxiliar o operador em suas decisões de compra e venda. A teoria fundamentando tais ferramentas é que os preços refletem não apenas a oferta e demanda em um determinado momento, mas também padrões comportamentais e psicológicos dos participantes do mercado (MURPHY, 1999). Assim, ao incorporar tais indicadores ao modelo *LSTM*, verificaremos se o uso dessas ferramentas técnicas, em conjunto com a modelagem das dependências temporais, resulta em previsões mais precisas e úteis para decisões de *trading* no Mini Índice Bovespa.

Para tal objetivo, faremos uso de uma arquitetura de rede neural chamada *LSTM, que é* uma variação das redes neurais recorrentes (RNNs) projetadas especificamente para lidar com a dificuldade que arquiteturas mais simples enfrentam ao tentar aprender padrões em dados sequenciais quando essas relações estão distantes no tempo, como em uma série temporal (STAUDEMEYER; MORRIS, 2019). Esta arquitetura supera essa limitação por possuir um mecanismo interno que funciona como uma memória que permite reter informações anteriores por longos períodos. Essa capacidade faz com que elas sejam eficazes em capturar dependências e padrões de longo prazo em dados temporais, como os movimentos de preços no mercado financeiro.

As seções a seguir abordarão o que se deseja que este estudo realize e as hipóteses assumidas; a teoria de mercado financeiro e redes neurais que envolve este trabalho; como os dados foram obtidos e processados e, por fim, os resultados esperados. O trabalho segue, a partir da Introdução, com os Objetivos, onde se descreve e aponta o que pretendemos realizar; Justificativa, onde se explica a escolha do Mini Índice e a arquitetura LSTM; Referencial Teórico, seção na qual apresenta e detalha aspectos relevantes do mercado financeiro, do mercado de futuros e os indicadores técnicos, e os principais conceitos sobre redes neurais e a arquitetura LSTM; Metodologia, onde detalha os dados obtidos, se explica a obtenção e pré-processamento como também apresenta as métricas de avaliação a serem utilizadas; Desenvolvimento da solução aborda a construção do modelo, com suas camadas e hiperparâmetros; Resultados obtidos é a seção onde detalhamos e comparamos as métricas de avaliação entre os modelos; e, por fim, resta as Conclusões onde

apontamos o que foi indicado no presente trabalho, assim como futuras direções de investigações possíveis.

2 OBJETIVOS

2.1 OBJETIVO GERAL

O objetivo deste trabalho é avaliar a capacidade preditiva das Redes Neurais Recorrentes do tipo *Long Short-Term Memory* (LSTM) na previsão dos preços de fechamento do Mini Índice do Bovespa, e averiguar qual é o impacto da adição dos indicadores técnicos sobre a qualidade das predições feitas pelo modelo.

2.2 OBJETIVOS ESPECÍFICOS

- Comparar o desempenho preditivo de modelos LSTM com diferentes variáveis, incluindo indicadores técnicos.
- Avaliar a capacidade preditiva de uma rede neural LSTM na previsão de retornos do Mini Índice Bovespa, utilizando métricas como o MAPE e MAE.
- Delinear o custo computacional necessário para a criação de um modelo desta natureza.
- Medir o impacto dos indicadores técnicos em um modelo preditivo.
- Observar a performance das redes neurais LSTM com dados de maior granularidade.

3 JUSTIFICATIVA

O presente trabalho buscar investigar a capacidade preditiva do modelo aplicado ao Mini Índice do Bovespa com os dados contendo intervalos de 60 minutos. Foi escolhido o Mini índice pois possui um alto volume de negociação, na casa das centenas de milhões, e liquidez também notável. Além disso, é um instrumento bastante popular no nosso mercado de futuro para operações de curto prazo e *hedging*.

Dentro do campo das redes neurais existem várias arquiteturas que podem ser utilizadas, contudo, a LSTM possui a capacidade de permitir que informações de eventos passados influenciem as previsões futuras através de três portas: esquecimento, de entrada e de saída. Em comparação com modelos estatísticos como ARIMA e modelos baseados em regressão, que exigem suposições sobre a linearidade e estacionaridade dos dados, a LSTM se destaca por aprender automaticamente relações não lineares. Isso torna a LSTM particularmente interessante para a previsão de séries temporais que podem possuir padrões que se repetem ao longo do tempo, sendo uma escolha equilibrada entre capacidade de modelagem temporal e eficiência computacional.

4 REFERENCIAL TEÓRICO

4.1 MERCADO FINANCEIRO

4.1.1 Mercado de futuros

O mercado de futuros desempenha um papel fundamental no sistema financeiro global, onde compradores e vendedores negociam contratos padronizados para a compra ou venda de um ativo, que pode ser uma *commodity*, moeda, índices de ações, em uma data futura específica, a um preço acordado hoje (BODIE; KANE; MARCUS, 2021). É um mercado bastante popular para especulação financeira, uma vez que é negociado um alto volume, existe liquidez suficiente para que os participantes realizem operações rápidas, muitas vezes abertas e fechadas no mesmo dia, em busca de lucros, se aproveitando de desequilíbrios nos preços (HULL, 2018).

4.1.2 Mini índice

O mini índice é um dos instrumentos oferecidos no mercado de futuros brasileiro, possibilitando aos investidores negociar a variação de pontos do Índice Bovespa (Ibovespa) para um período futuro, sem precisar adquirir ou vender os ativos individuais que formam o índice (B3, 2024). Ao contrário do contrato padrão de índice futuro, o mini índice possui um valor menor por ponto de variação do Ibovespa,

tornando-o acessível a investidores individuais e de menor porte. Cada ponto do mini índice corresponde a uma fração do valor do contrato futuro padrão.

A negociação do mini índice é regulamentada pela B3, com regras claras de liquidação financeira e mecanismos para garantir a segurança e integridade das transações. Os contratos de mini índice têm ciclos de vencimento a cada dois meses, ocorrendo nos meses pares do ano, como fevereiro, abril, junho, agosto, outubro e dezembro. O último dia de negociação geralmente é na quarta-feira mais próxima ao dia 15 do mês de vencimento, com a data oficial de vencimento sendo o dia seguinte. Neste dia, ocorre a liquidação financeira dos contratos, onde os lucros e prejuízos são ajustados nas contas dos investidores.

4.2 ANÁLISE TÉCNICA

A análise técnica é uma metodologia utilizada para prever a direção futura dos preços dos ativos financeiros com base no estudo de dados históricos de preços e volumes de negociação (MURPHY, 1999). Essa abordagem tem como fundamenta-se na afirmação que o comportamento passado dos preços possui poder preditivo do movimento futuro, pois os mercados financeiros tendem a seguir certos padrões e tendências.

No início do século XX que a análise técnica começou a ganhar forma como uma disciplina estruturada, especialmente nos Estados Unidos (PRING, 2014). Com a popularização dos computadores na década de 70, a análise técnica evoluiu significativamente, permitindo que grandes quantidades de dados fossem processadas e analisadas rapidamente. Isso possibilitou o desenvolvimento de indicadores mais sofisticados e o surgimento do *trading* algorítmico. Com o advento da internet e a acessibilidade a plataformas de negociação online, a análise técnica se popularizou entre investidores de varejo ao redor do mundo.

Contudo, não é uma abordagem livre de críticas. Em primeiro lugar, a análise técnica e seus indicadores baseiam-se na premissa de que padrões passados de preços e volumes podem prever movimentos futuros, porém, essa abordagem carece de uma base teórica sólida, o que coloca em dúvida sua eficácia como ferramenta preditiva (SILVA; MEDEIROS, 2017). Além disso, estudos mostram que esses

métodos apresentam falta de consistência estatística, uma vez que, no longo prazo, estratégias baseadas em indicadores técnicos não superam consistentemente o mercado, sugerindo que muitos dos padrões identificados podem ser resultado de aleatoriedade (CORNETTA, 2011). Outra limitação significativa é informações fundamentais como balanços financeiros, políticas monetárias e eventos macroeconômicos, que podem influenciar diretamente o valor dos ativos, são completamente ignorados pelos indicadores, o que pode levar a decisões imprecisas quando utilizados isoladamente (RANKIA BRASIL, 2023).

4.3 INDICADORES TÉCNICOS

Os indicadores técnicos surgiram no início do século XX, com pioneiros como Charles Dow e Richard Wyckoff, que buscavam métodos para prever movimentos de mercado através da análise gráfica (PRING, 2014). Esses indicadores são usados para encontrar tendências, medir a força de movimentos de preços, e detectar condições de sobrecompra ou sobrevenda, sendo aplicados tanto em operações de curto prazo, como no *day trade,* onde a posição é aberta e fechada no mesmo dia, quanto em estratégias de longo prazo. Apesar de haver estudos questionando a credibilidade desses indicadores, ainda são bastante populares atualmente (MALKIEL, 2019). Os indicadores utilizados no segundo modelo são: MACD, Bandas de Bollinger, RSI, OBV, SMA e EMA.

4.3.1 MACD

O MACD, criado por Gerald Appel em 1979, é um dos indicadores técnicos mais populares na análise técnica (APPEL, 2005). Ele consiste em duas linhas principais: a linha MACD, obtida pela subtração de duas médias móveis exponenciais, uma de 12 e outra de 26 períodos, e a linha de sinal, que também é uma média móvel exponencial, mas da linha MACD e usualmente possui 9 períodos. Quando a linha MACD cruza acima da linha de sinal, é considerado um sinal de compra, enquanto um cruzamento abaixo da linha de sinal é um sinal de venda. O MACD também inclui um histograma que mostra a diferença entre as duas linhas, ajudando a identificar a força de uma tendência.

$$Linha\ MACD = EMA\ de\ 12\ períodos\ -\ EMA\ de\ 26\ períodos$$

$$Linha\ de\ Sinal = EMA\ de\ 9\ períodos\ da\ Linha\ MACD$$

$$Histograma = Linha\ MACD\ -\ Linha\ de\ Sinal$$

4.3.2 BANDAS DE BOLLINGER

Criadas no início dos anos 1980 por John Bollinger, as Bandas de Bollinger são um indicador que mede da intensidade e frequência das variações nos preços do mercado e são compostas por três linhas: uma média móvel simples (SMA) no centro, e duas bandas externas, que são desvios padrão da SMA (BOLLINGER, 2001). Quando os preços se movem para perto da banda superior, obtida pela média móvel somada ao desvio padrão multiplicado por dois, é possível que o ativo esteja sobrecomprado; quando se aproximam da banda inferior, obtida pela média móvel menos o desvio padrão multiplicado por dois, pode estar sobrevendido. As Bandas de Bollinger são frequentemente usadas para identificar potenciais reversões de tendência ou para confirmar a direção do mercado em momentos de alta volatilidade.

$$SMA = \frac{\sum_{i=1}^{n} \mathrm{Pr}\, e\, ço\ de\ fechamento}{n}$$

$$DP = \sqrt{\frac{\sum_{i=1}^{n} (\mathrm{Pr}\, eço\ de\ fechamento_i - SMA)^2}{n}}$$

$$Banda\ Superior\ =\ SMA\ +\ (2\ \times\ DP)$$

$$Banda\ Inferior\ =\ SMA\ -\ (2\ \times\ DP)$$

4.3.3 RSI

Introduzido em 1978, o RSI foi criado por J. Welles Wilder Jr. e é um oscilador que mede a velocidade e a mudança dos movimentos de preço, oscilando entre 0 e 100 (WILDER, 1978). Como as Bandas de Bollinger, ele é usado para identificar condições de sobrecompra ou sobrevenda em um ativo. Um valor acima de 70 geralmente indica que o ativo está sobrecomprado, sugerindo uma possível reversão para baixo, enquanto um RSI abaixo de 30 indica sobrevenda, sugerindo uma possível reversão para cima. O RSI também pode ser usado para identificar divergências entre

a direção do preço e o indicador, o que pode sinalizar uma mudança iminente na tendência.

$$RS = \frac{Média\ dos\ Ganhos}{Média\ das\ Perdas}$$

$$RSI = 100 - \left(\frac{100}{1 + RS}\right)$$

4.3.4 OBV

O OBV é um indicador de momento que relaciona o volume de negociação com os movimentos de preço. A teoria por trás do OBV é que o volume precede o preço; portanto, mudanças no OBV podem sinalizar potenciais movimentos futuros do preço (GRANVILLE, 1963). Quando o OBV está em tendência de alta, pode indicar que o dinheiro está entrando no ativo, sugerindo uma possível alta no preço. Se o OBV estiver caindo, pode indicar que o dinheiro está saindo, o que pode prever uma queda no preço.

Se o preço do fechamento do período for maior que o preço de fechamento do período anterior, então somamos o valor do OBV anterior ao volume:

$$OBV_t = OBV_{t-1} + Volume_t$$

Se o preço do fechamento for menor, então subtraímos os valores:

$$OBV_t = OBV_{t-1} - Volume_t$$

Caso seja o mesmo valor, então o OBV permanecerá constante:

$$OBV_t = OBV_{t-1}$$

4.3.5 SMA

A Média Móvel Simples (SMA - *Simple Moving Average*) é indicador bastante comum na análise de séries temporais, especialmente no mercado financeiro, e relativamente simples de se calcular, sendo a média aritmética de um conjunto de valores de preços ou dados financeiros ao longo de um período específico. Sua principal função é suavizar flutuações de curto prazo e identificar tendências em um conjunto de dados (SOUZA; SILVA, 2021).

A SMA é amplamente utilizada por analistas técnicos para identificar tendências de alta e baixa, atuando como um suporte visual para a tomada de decisões. Quando o preço de um ativo está acima da sua média móvel, pode indicar uma tendência de alta; caso contrário, pode sinalizar uma tendência de queda.

4.3.6 EMA

A Média Móvel Exponencial (EMA - *Exponential Moving Average*) é similar a SMA, mas atribui um peso maior aos dados mais recentes, isso faz com que ela mais responda mais rapidamente às variações do mercado, reduzindo o atraso na identificação de tendências (MURPHY, 1999). Por outro lado, ela também pode gerar sinais falsos de reversão de tendência. Matematicamente, a EMA é calculada por meio da seguinte equação:

$$EMA_t = P_t \times \alpha + EMA_{t-1} \times (1 - \alpha)$$

Onde P_t representa o preço do ativo no instante t; α é o fator de suavização, calculado como $\frac{2}{n+1}$, sendo n o número de períodos considerados; EMA_{t-1} é a EMA do período anterior.

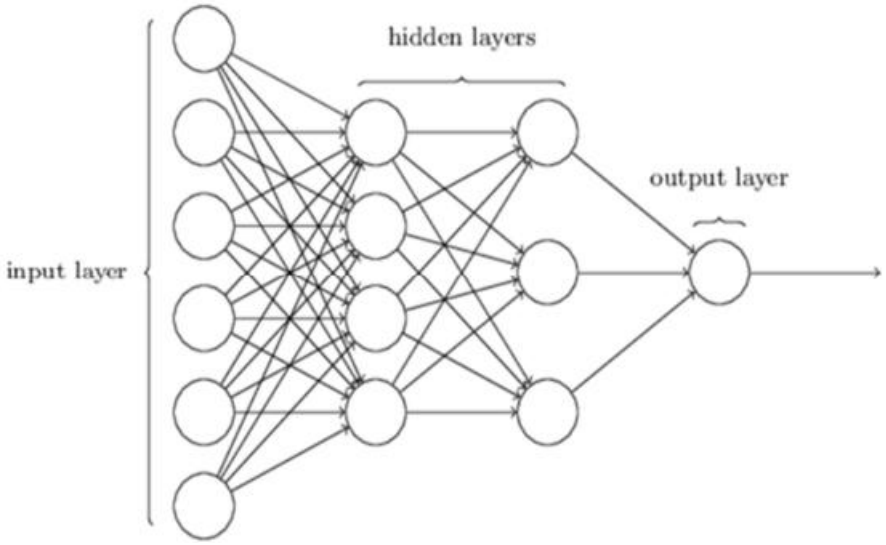
4.4 REDES NEURAIS

4.4.1 FUNCIONAMENTO GERAL

As Redes Neurais Artificiais (ANNs) têm conquistado um papel cada vez mais relevante no campo da inteligência artificial e do aprendizado de máquina, especialmente com a popularização de LLMs (*Large Language Models*), como o Chat GPT, da empresa OPENAI. Inspiradas no funcionamento do cérebro humano, as ANNs são compostas por unidades interconectadas, também chamadas de "neurônios", que trabalham em conjunto para processar informações e resolver problemas complexos (MCCULLOCH; PITTS, 1990), como reconhecimento de voz, visão computacional e jogos, sendo especialmente úteis para processar dados não estruturados, ou seja, que não seguem um formato ou esquema predefinido. A Figura

1 mostra uma rede neural genérica, com uma camada de entrada, camadas ocultas de processamento e uma camada de saída.

Figura 1 – Rede neural artificial



Fonte: Thakur e Konde (2021)

O componente básico de uma ANN é o neurônio artificial, que recebe uma ou mais entradas, aplica uma soma ponderada dessas entradas e, em seguida, utiliza uma função de ativação para gerar uma saída. A função de ativação é responsável por introduzir não linearidades no modelo, permitindo que a rede aprenda padrões complexos. As mais comuns são: Sigmoid, ReLU (*Rectified Linear Unit*), Tanh (Tangente Hiperbólica) e Softmax (GOODFELLOW; BENGIO; COURVILLE, 2016).

As camadas ocultas possibilitam que a rede aprenda a associar entradas a saídas por meio de um processo iterativo de ajuste de pesos e vieses. Esse aprendizado geralmente ocorre através da retropropagação, que emprega o gradiente do erro em relação aos pesos da rede para ajustá-los, com o objetivo de minimizar a função de custo, a qual mede a discrepância entre as previsões e os valores reais.

O poder das ANNs, especialmente das redes neurais profundas, está na sua capacidade de aprender representações hierárquicas dos dados, onde camadas superiores capturam abstrações de alto nível a partir de representações mais básicas aprendidas nas camadas inferiores.

Apesar de terem sido concebidos na década de 1940, foi em 1980 que as ANNs voltaram a ganhar atenção, uma vez que o algoritmo de retropropagação, que permite que redes neurais com múltiplas camadas (chamadas de redes neurais profundas) ajustem seus pesos de forma eficiente, resolvendo problemas complexos que antes eram intratáveis, foi proposto por Rumelhart, Hinton e Williams (RUMELHART; HINTON; WILLIAMS, 1986). Além disso, com o poder computacional cada vez maior e mais acessível, somado a enorme quantidade de dados que a internet nos proporcionou, tem crescido cada vez mais a popularidade deste método.

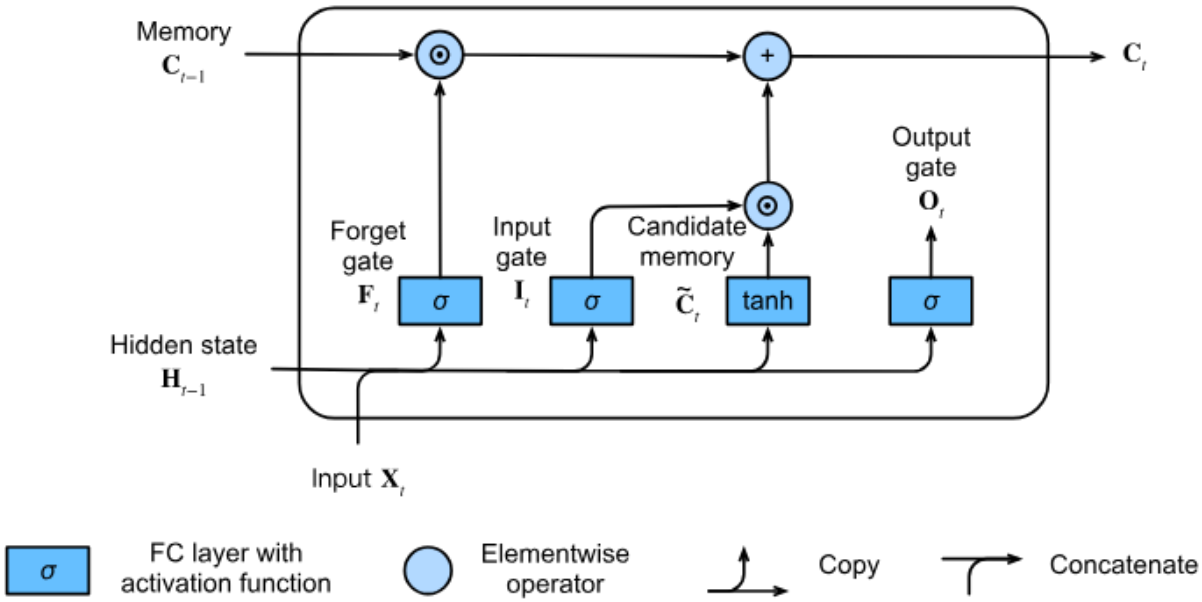
4.4.2 ARQUITETURA LSTM

Com o avanço das redes neurais e seu uso em cenários cada vez mais específicos, nasceram outras formas de organizar os “neurônios” artificiais a fim de superar a performance dos modelos mais simples. Assim, foram desenvolvidas Redes Convolucionais (CNNs), bastante populares no processamento de imagens; Redes Recorrentes (RNNs), que se destacam na análise de dados sequenciais, como texto ou séries temporais; e a Long Short-Term Memory (LSTM), uma variação das RNNs projetada para lidar com dados sequenciais de longo prazo (HOCHREITER; SCHMIDHUBER, 1997).

Uma LSTM é composta por unidades básicas que armazenam e gerenciam informações ao longo do tempo, também chamadas de células de memória, que regulam o fluxo de informações por meio de três componentes, que podem ser chamados de “portas”, principais: a porta de entrada, a porta de esquecimento e a porta de saída (STAUDEMEYER; MORRIS, 2019).

Cada uma dessas portas desempenha um papel crucial na manutenção e atualização das informações ao longo do tempo. Ao longo de uma sequência de dados, dois estados interagem entre si para processar informações sequenciais: Estado da Célula (C_t), que armazena informações de longo prazo e serve como uma “memória” de longo prazo, onde informações importantes são armazenadas e acessadas; e o Estado Oculto (h_t), que representa as informações de curto prazo e serve como saída da célula, age como um “resumo” das informações mais relevantes para a saída atual, ou seja, desempenha o papel de memória de curto prazo.

Figura 2 – Arquitetura LSTM



Fonte: Zhang et al. (2021)

5 METODOLOGIA

5.1 DADOS UTILIZADOS

Os dados utilizados são os preços de abertura, fechamento, máxima e mínima, volume, dia e data da série contínua do Mini Índice no intervalo de quinze minutos. Foram obtidos da plataforma MetaTrader 5, e compreendem do dia 01/01/2022 até 31/12/2024. A esses dados, foram adicionados os indicadores técnicos descritos neste trabalho com a biblioteca *technical analysis* para a linguagem de programação Python.

5.2 PRÉ-PROCESSAMENTO DOS DADOS

Após a coleta dos dados manualmente através da plataforma MetaTrader, foi feita uma simples limpeza dos dados envolvendo: retiradas de símbolos dos nomes das colunas e a retirada das colunas “<TICKVOL>” e “<SPREAD>” da base, por serem irrelevantes para o propósito do trabalho.

Uma importante etapa ao fazermos uso de redes neurais é normalizarmos os dados, uma vez que essa etapa melhora o desempenho e a eficiência do treinamento

do modelo e facilita a minimização da função de perda. O método utilizado foi o min-max, que reescala os valores de tal forma que fiquem entre 0 e 1 (JAIN; DUIN; MAO, 2000).

$$X_{normalizado} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Onde X é a variável que será normalizada, e X_{min} e X_{max} são seus valores mínimo e máximo. Além dessa etapa, também foram criadas janelas temporais, que são conjuntos de dados contendo uma sequência de valores passados, com 16 valores.

O MinMaxScaler é adequado para redes LSTMs porque ele não altera a forma da distribuição dos dados, apenas os reescala para o intervalo entre 0 e 1, isso preserva a preserva a escala relativa dos valores ao mesmo tempo que reduz a amplitude das entradas da LSTM.

5.3 MÉTRICAS DE AVALIAÇÃO

No contexto de avaliação de modelos de previsão de séries temporais, existem diversas métricas para determinarmos a qualidade de um modelo. Para esse estudo, foi utilizado o erro percentual absoluto médio, que tem como sigla MAPE (*Mean Absolute Percentage Error*) que representa a média do erro percentual absoluto entre os valores previstos e os valores reais, portanto, quanto menor o MAPE, melhor o modelo. Sua fórmula é a seguinte:

$$M = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\mathrm{A}i\ -\ Fi}{Ai}\right|$$

Sendo M o erro percentual absoluto médio, n é o número total de observações, Ai é o valor real na i - ésima posição, Fi é o valor previsto pelo modelo na i - ésima posição e $|Ai - Fi|$ é o erro absoluto.

Além disso, podemos diferenciar três medidas de MAPE: o MAPE padrão, ou geral, indica o erro percentual médio considerando todos os dados (treino e teste); o MAPE treino representa o erro nos dados de treinamento; e o MAPE teste avalia o modelo nos dados de teste, ou seja, aqueles que não são vistos durante o treinamento. Essa distinção é importante pois nos permite avaliar como o modelo lida com dados não vistos antes.

Essa métrica de avaliação foi utilizada por ser uma métrica intuitiva e interpretável e os valores não estão próximos de zero, o que, conforme explicitado por Hyndman e Koehler (2006), pode ser problemático, uma vez que a fórmula envolve divisão pelo número real.

Uma vez que o MAPE indica os erros em termos percentual, também incluímos o MAE (*Mean Absolute Error*) que representa a média do erro absoluto entre os valores previstos e os valores reais na mesma unidade dos dados reais. Quanto menor o MAE, melhor o modelo. Também foi feito com o MAPE, fizemos a distinção entre o MAE padrão, que indica o erro considerando os dados de teste e treino; MAE Treino, representando o erro nos dados de treinamento; e o MAE Teste, que avalia o modelo somente nos dados de teste.

6 DESENVOLVIMENTO DA SOLUÇÃO

Foram criados 4 modelos LSTM com os mesmos hiperparâmetros. O modelo 1 contém apenas o preço de fechamento, o modelo 2 contém os preços de abertura, máximo, mínimo e fechamento, o modelo 3 contém além dos preços de abertura, máximo, mínimo e fechamento, também foi incluída a média móvel exponencial de 14 períodos e, finalmente, o modelo 4 acrescenta, em relação ao modelo 3, todos os indicadores técnicos listados anteriormente.

Para todas essas variáveis foram utilizados os 16 últimos valores de cada parâmetro como entrada através de janelas temporais do tipo deslizante. Assim, as informações dos últimos 16 períodos são usadas como variáveis para alimentar o modelo. Isso permite a identificação de padrões e sazonalidades conforme demonstrado por Perea e Harer (2013) e é essencial para capturar dependências temporais e melhorar a precisão das previsões. Tal janela foi aplicada a todas as

variáveis. Um resumo dos modelos e suas variáveis se encontram na Tabela 1. Todos modelos tem como saída o preço de fechamento do próximo período.

Tabela 1 – Modelos e variáveis utilizadas

Modelo	Variáveis Utilizadas nos Modelos
1	Close
2	Close, Open, High, Low
3	Close, Open, High, Low, EMA
4	Close, Open, Max, Min, MACD, BB_upper, BB_lower, RSI, OBV, SMA, EMA

Fonte: Elaborado pelo autor (2025)

As variáveis foram obtidas tanto a partir do MetaTrader quanto da biblioteca *Technical Analysis:*

Tabela 2 - Descrição e fonte das variáveis utilizadas

Variável	Descrição	Fonte
Close	Preço de fechamento	MetaTrader
Open	Preço de abertura	MetaTrader
Max	Maior preço alcançado entre a abertura e o fechamento	MetaTrader
Min	Menor preço alcançado entre a abertura e o fechamento	MetaTrader
Vol	Volume de negociação entre a abertura e o fechamento	MetaTrader
MACD	Indicador *MACD*	*Technical Analysis*
BB_upper	Limite superior da Banda de Bolliger	*Technical Analysis*
BB_lower	Limite inferior da Banda de Bolliger	*Technical Analysis*

RSI	Linha do indicador RSI	*Technical Analysis*
OBV	Linha do indicador *OBV*	*Technical Analysis*
SMA	Média móvel simples	*Technical Analysis*
EMA	Média móvel exponencial	*Technical Analysis*

Fonte: Elaborado pelo autor (2025)

Em relação ao modelo LSTM, foram utilizadas 3 camadas: a primeira com 100 neurônios, a segunda com 50 e a terceira e última, com apenas uma unidade, que é o a previsão feita pelo modelo. Dessa forma, procurou-se equilibrar a complexidade dos dados sem aumentar excessivamente a complexidade do modelo, evitando assim o risco de *overfitting*. Além disso, depois da primeira e segunda camada de neurônios, foram adicionadas camadas Leaky ReLU. Essas camadas, introduzidas por Maas, Hannun e Ng (2013) são uma variação da função de ativação ReLU (*Rectified Linear Unit*) utilizadas para evitar o problema do "neurônio morto", que acontece quando as saídas de alguns neurônios são sempre zero, o que pode prejudicar o aprendizado do modelo.

O algoritmo de otimização escolhido foi o *Adam,* desenvolvido por Kingma e Ba (2014) que é amplamente utilizado em redes LSTM devido à sua rápida convergência, estabilidade e ajuste adaptativo da taxa de aprendizado. Ele combina as vantagens do *Momentum*, que suaviza a descida do gradiente, e do *RMSprop*, que ajusta a taxa de aprendizado para cada parâmetro, tornando-o altamente eficiente em dados ruidosos e não estacionários.

Em ambas as redes foi utilizado como algoritmo de perda o erro médio quadrático (*MSELoss*) que é a média dos erros quadráticos entre previsões e valores reais, penalizando mais os erros maiores e suavizando erros menores.

7 RESULTADOS OBTIDOS

Para facilitar a análise chamaremos o modelo 1 de 'modelo base', o modelo 2 de 'modelo que inclui os dados OHLC' (numa referência aos dados de preços de abertura, máximo, mínimo e fechamento), o modelo 3 de 'modelo que inclui os dados OHLC e a média móvel exponencial de 14 períodos' e o modelo 4 de 'modelo que inclui os dados OHLC e todos os indicadores citados'. O modelo base obteve um MAPE padrão de 1.3381%, o modelo que inclui os dados OHLC obteve um MAPE padrão de 1.0657%, o modelo que inclui os dados OHLC e a média móvel exponencial de 14 períodos obteve um MAPE padrão de 1.1358%, enquanto no modelo que inclui os dados OHLC e todos os indicadores citados foi de 0.6687%. Isso sugere que, em geral, o modelo 4 teve um desempenho melhor em termos de precisão percentual.

Em relação ao MAPE treino, o modelo base obteve 1.4407%, o modelo que inclui os dados OHLC obteve um MAPE treino de 1.0432%, o terceiro modelo teve o MAPE treino em 1.1620%, e o modelo que inclui os dados OHLC e todos os indicadores citados 0.5216%, mais uma vez foi indicado que o modelo 4 apresentou melhor desempenho. Para o MAPE teste, o modelo base obteve um MAPE de 0.8597%, enquanto o modelo que inclui os dados OHLC obteve 1.1706%, o modelo inclui os dados OHLC e a média móvel exponencial de 14 períodos obteve um MAPE de teste de 1.0131% e o modelo que inclui os dados OHLC e todos os indicadores citados obteve um MAPE teste de 1.3570%, essa discrepância indica que, com dados não vistos, o modelo base foi avaliado como superior ao modelo que inclui os dados OHLC e os indicadores, o que pode indicar que este modelo possui uma capacidade de generalização maior e aponta a possibilidade de ter ocorrido *overfitting* no modelo mais complexo. Na tabela a seguir estão descritos os resultados:

Tabela 3 – Resultados dos modelos expresso pelo MAPE

Modelo	MAPE geral	MAPE treino	MAPE teste
1	1.3381%	1.4407%	0.8597%
2	1.0657%	1.0432%	1.1706%
3	1.1358%	1.1620%	1.0131%
4	0.6687%	0.5216%	1.3570%

Fonte: Elaborado pelo autor (2025)

Nos MAE geral, o modelo base obteve 1517.55, o modelo que inclui os dados OHLC obteve um MAE geral de 1251.06, o modelo que inclui os dados OHLC obteve 1328.63, enquanto o quarto ficou com 819.78. Para o treino, o modelo base obteve 1596.93, o modelo que inclui os dados OHLC obteve um MAE de treino de 1183.48, o modelo que inclui os dados OHLC ficou com 1324.31, enquanto o modelo que inclui os dados OHLC e todos os indicadores citados 606.24. E entre os dados não vistos antes (teste) o MAE ficou em 1147.18 para o modelo base, 1567.28 para o modelo que inclui os dados OHLC, 1348.83 para o modelo que inclui os dados OHLC e a média móvel exponencial de 14 períodos e 1818.95 para o modelo que inclui os dados OHLC e todos os indicadores citados.

Portanto, o modelo que inclui os dados OHLC e indicadores técnicos apresentou desempenho melhor que os outros modelos tanto em termos de precisão percentual (MAPE) quanto em termos de erro absoluto (MAE) para os dados de treinamento, mas não superou os outros modelos nos dados não vistos antes, isso indica que o modelo mais complexo não foi capaz de generalizar os dados de forma adequada. Assim, o modelo mais complexo obteve métricas melhores, mas é necessário atenção no uso deste modelo em dados não vistos antes.

Tabela 4 – Resultados dos modelos expressos pelo MAE

Modelo	MAE geral	MAE treino	MAE teste
1	1517.55	1596.93	1147.19
2	1251.06	1183.48	1567.28
3	1328.63	1324.31	1348.83
4	819.78	606.24	1818.95

Fonte: Elaborado pelo autor (2025)

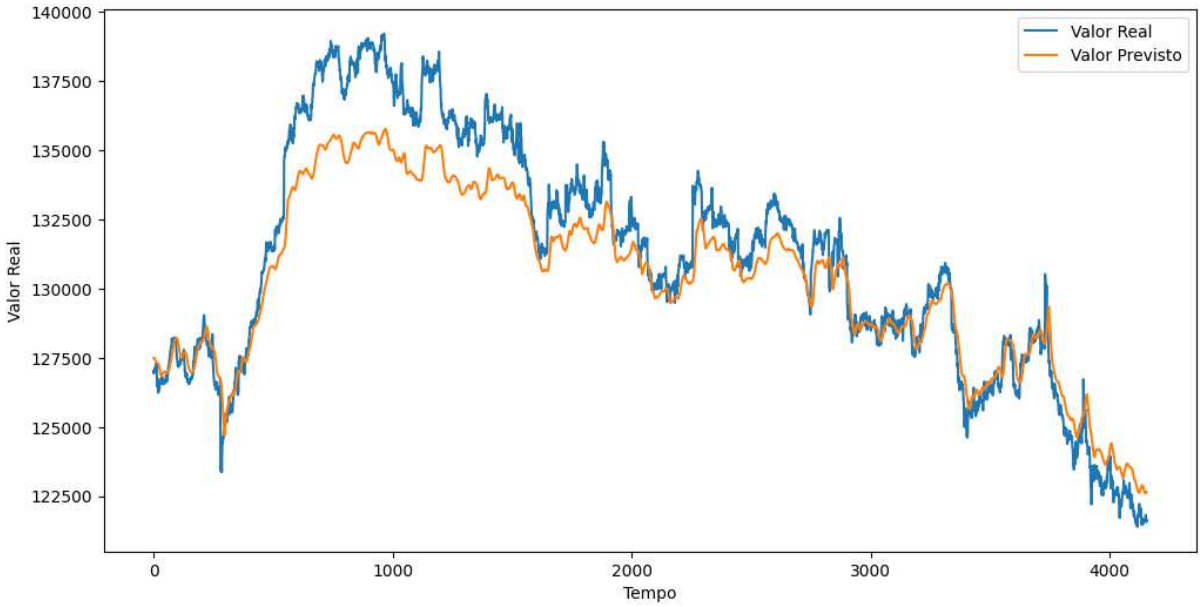


Gráfico 1 – Comparativo entre previsões e valores reais do modelo base

Fonte: Elaborado pelo autor (2025)

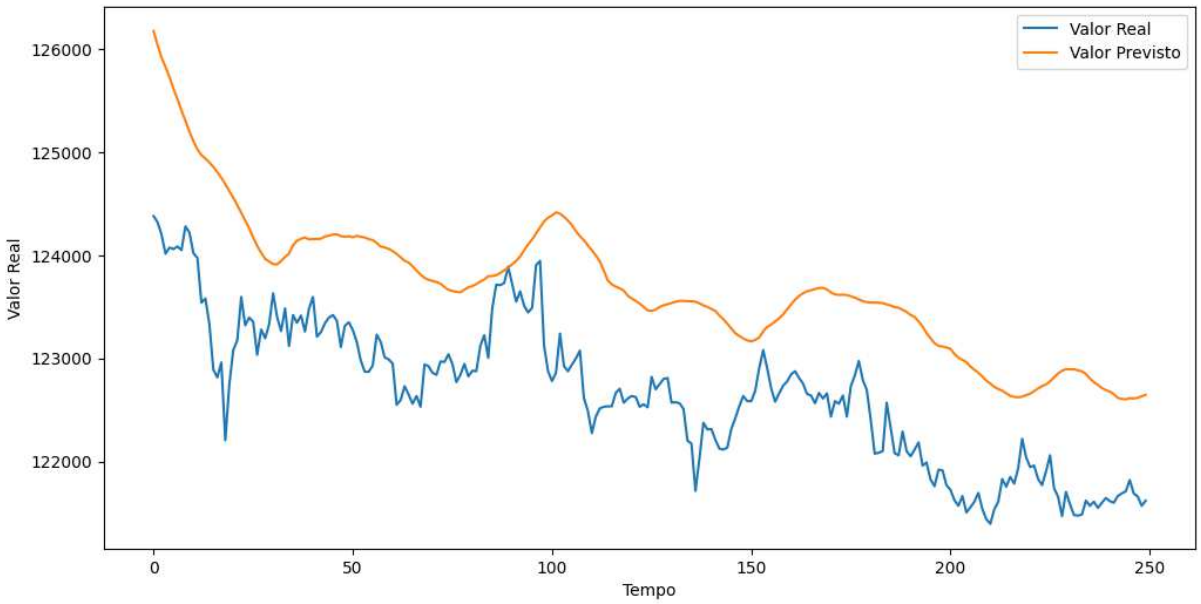


Gráfico 2 – Comparativo entre previsões e valores reais do modelo base com os últimos 250 dados

Fonte: Elaborado pelo autor (2025)

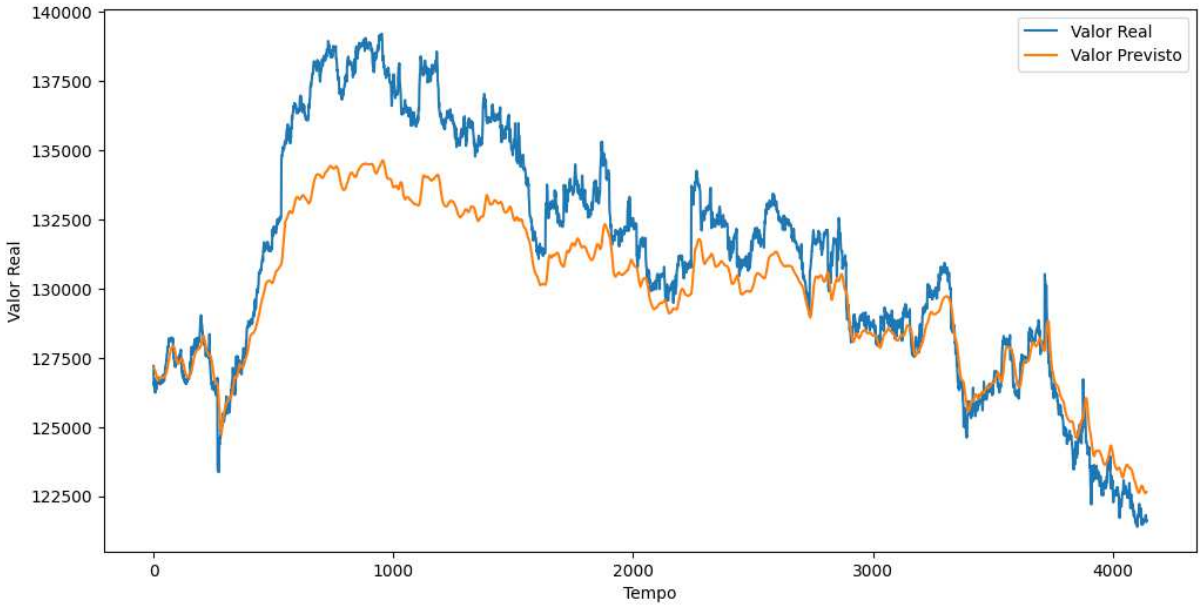


Gráfico 3 – Comparativo entre previsões e valores reais do modelo que inclui OHLC

Fonte: Elaborado pelo autor (2025)

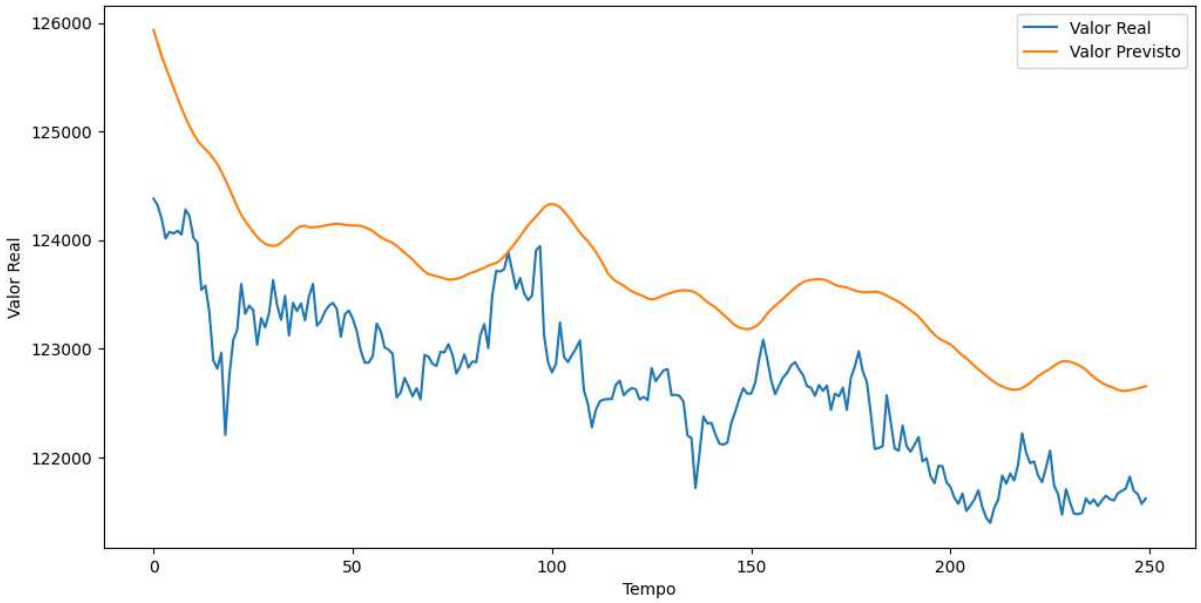


Gráfico 4 – Comparativo entre previsões e valores reais do modelo que inclui OHLC com os últimos 250 dados

Fonte: Elaborado pelo autor (2025)

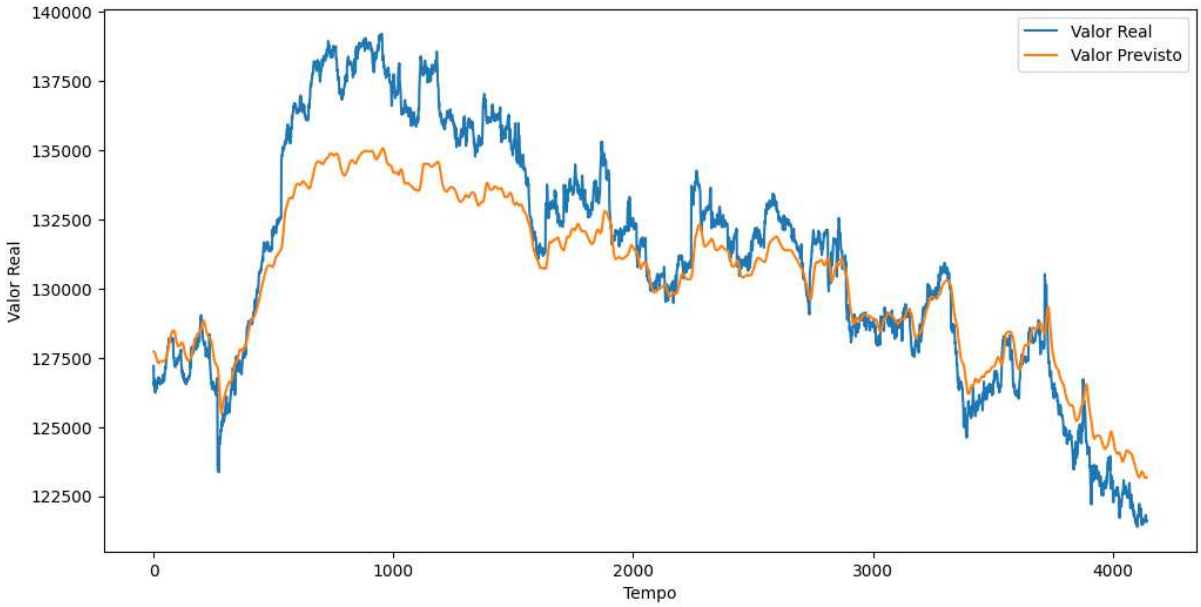


Gráfico 5 – Comparativo entre previsões e valores reais do modelo que inclui os dados OHLC e a média móvel exponencial de 14 períodos

Fonte: Elaborado pelo autor (2025)

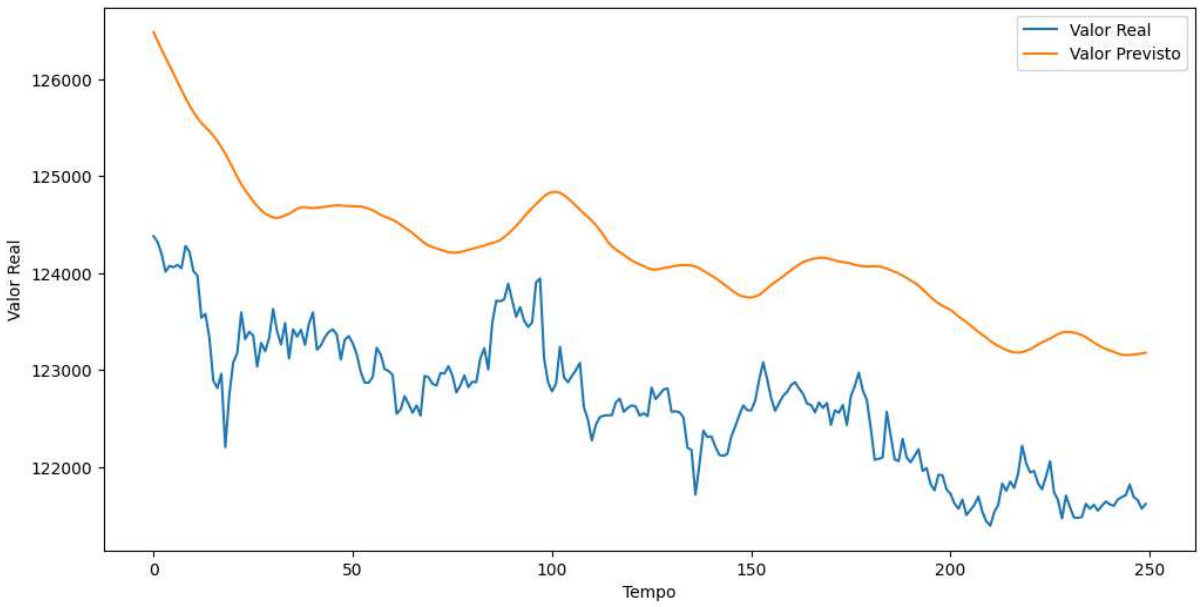


Gráfico 6 – Comparativo entre previsões e valores reais do modelo que inclui os dados OHLC e a média móvel exponencial de 14 períodos com os últimos 250 dados

Fonte: Elaborado pelo autor

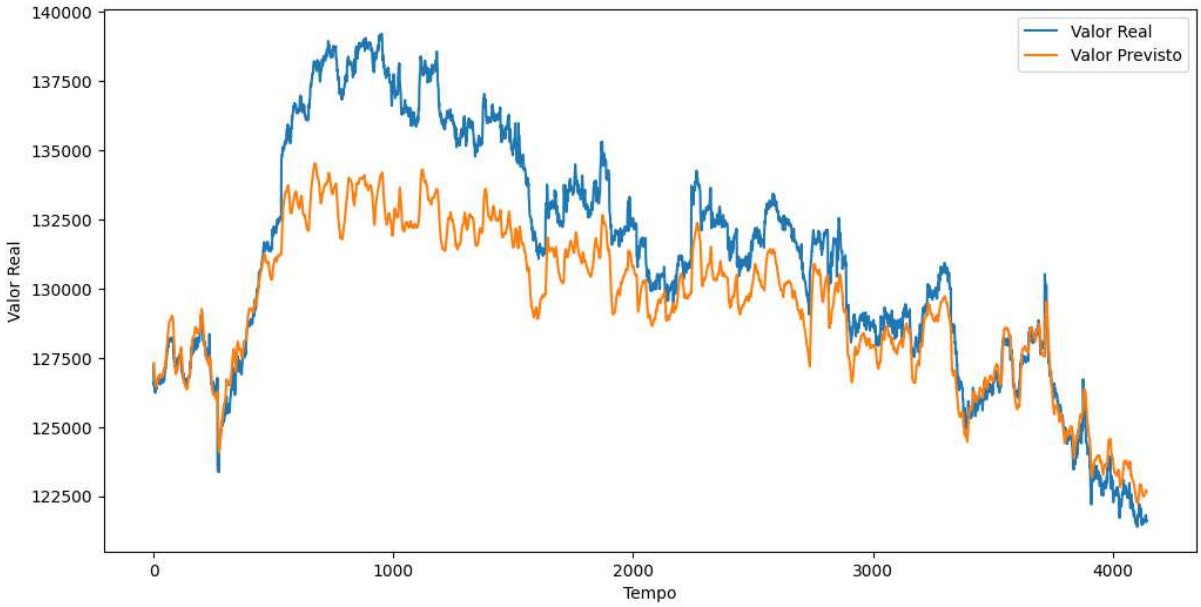


Gráfico 7 – Comparativo entre previsões e valores reais do modelo que inclui os dados OHLC e todos os indicadores citados

Fonte: Elaborado pelo autor (2025)

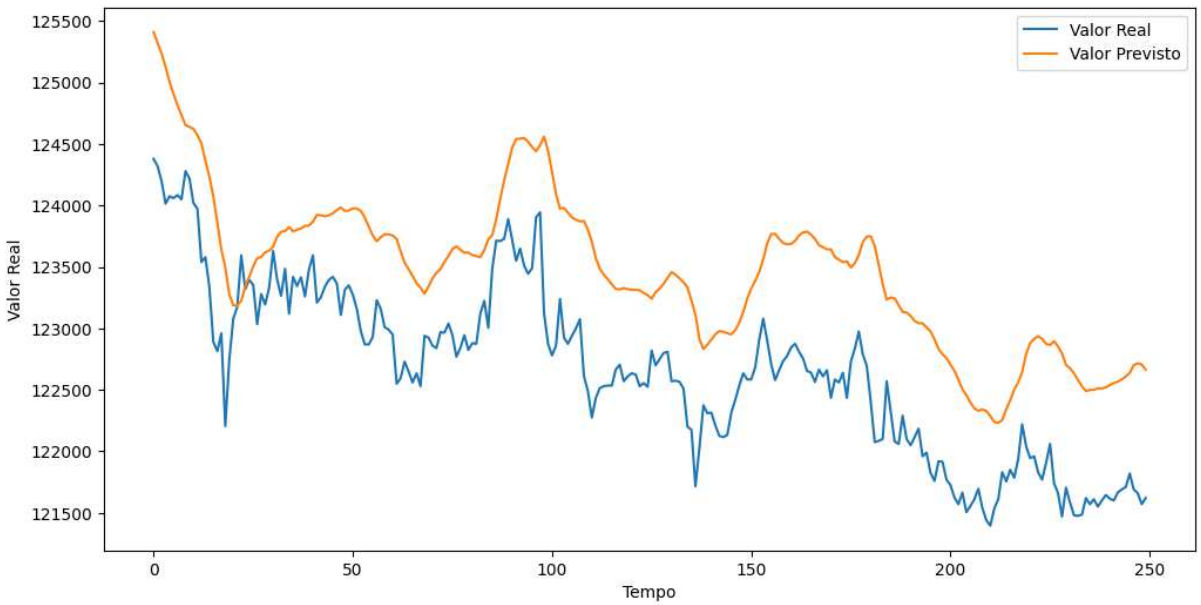


Gráfico 8 – Comparativo entre previsões e valores reais do modelo que inclui os dados OHLC e todos os indicadores citados com os últimos 250 dados

Fonte: Elaborado pelo autor (2025)

É necessário esclarecer que o MAPE é tão baixo por conta dos grandes valores que a série histórica utilizada possui. O MAPE aqui, portanto, nos indica apenas qual modelo é superior ao outro, e no caso de uma base de dados onde o valor médio é de aproximadamente 140 mil, podemos observar que o modelo não ostenta uma performance tão elevada quanto o baixo valor da medida de erro pode sugerir inicialmente.

8 CONCLUSÕES

Avaliando os 4 modelos, as métricas indicam que o modelo que faz uso das informações de preço e indicadores técnicos foi superior aos outros. Contudo, a diferença entre as métricas de avaliação de treino e de teste indicam que o modelo não generaliza de forma adequada. Por outro lado, o MAPE e MAE teste do modelo apenas com o preço de fechamento, indicam que ele foi capaz de generalizar de forma mais adequada. O modelo com as informações de preço e a média móvel exponencial de 14 períodos não teve um resultado interessante no geral.

Para estudos futuros, pode ser interessante incluir mais dados e montar modelos mais complexos, ou seja, com mais camadas ou mais neurônios ou tentar diferente combinações de indicadores técnicos. O desafio de ampliar a base de dados é incluir informações distorcidas pelo fenômeno extremamente raro que foi a COVID-19. Se o modelo assumir que aquele comportamento dos preços é normal, então suas previsões poderão ser gravemente penalizadas.

REFERÊNCIAS

APPEL, Gerald. **Technical Analysis: Power Tools for Active Investors**. 1. ed. New York: Financial Times Press, 2005.

BODIE, Z.; KANE, A.; MARCUS, A. J. **Investments**. 12. ed. New York, Ny: Mcgraw-Hill Education, 2021

BOLLINGER, John. **Bollinger on Bollinger Bands**. 1. ed. New York: McGraw-Hill, 2001

B3. **Futuro Mini de Ibovespa**. Disponível em: https://www.b3.com.br/pt_br/produtos-e-servicos/negociacao/renda-variavel/futuro-mini-de-ibovespa.htm. Acesso em: 18 set. 2024.

COMMODITY FUTURES TRADING COMMISSION (CFTC). **A Primer on Artificial Intelligence in Financial Markets**. Washington, D.C.: LabCFTC, 2019. Disponível em: https://www.cftc.gov. Acesso em: 18 set. 2024.

CORNETTA, Luiz Antonio. A insuficiência da análise técnica na composição de carteiras segundo a teoria dos mercados eficientes. 2011. Dissertação (Mestrado em Economia) – Universidade Estadual de Campinas, Campinas, 2011. Disponível em: https://repositorio.unicamp.br/Busca/Download?codigoArquivo=509977. Acesso em: 14 fev. 2025.

GHOSH, P.; NEUFELD, A.; SAHOO, J. K. Forecasting directional movements of stock prices for intraday trading using LSTM and random forests. **Finance Research Letters**, p. 102280, jul. 2021.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, Massachusetts: The Mit Press, 2016.

GRANVILLE, Joseph E. **New Key to Stock Market Profits**. 1. ed. Englewood Cliffs, N.J.: Prentice-Hall, 1963.

HAWKINS, J. ***The Rise of Computerized High Frequency*** *Trading*. Duke Law & Technology Review, v. 11, 2012. Disponível em: https://scholarship.law.duke.edu/cgi/viewcontent.cgi?article=1211&context=dltr.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long short-term memory. *Neural Computation*, v. 9, n. 8, p. 1735-1780, 1997

HULL, John C. *Options, futures, and other derivatives.* 10. ed. Boston: Pearson Education, 2018.

HYNDMAN, Rob J.; ATHANASOPOULOS, George. *Forecasting: principles and practice.* 2. ed. Melbourne: OTexts, 2018.

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

JAIN, Anil K.; DUIN, Robert P. W.; MAO, Jianchang. **Statistical Pattern Recognition: A Review**. IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 22, n. 1, p. 4-37, 2000

KINGMA, Diederik P.; BA, Jimmy. **Adam: a method for stochastic optimization**. arXiv, 2014. Disponível em: https://arxiv.org/abs/1412.6980. Acesso em: 15 fev. 2025.

LUDERMIR, T. B. Inteligência Artificial e Aprendizado de Máquina: estado atual e tendências. **Estudos Avançados**, v. 35, n. 101, p. 85–94, abr. 2021.

MAKRIDAKIS, Spyros; WHEELWRIGHT, Steven C.; HYNDMAN, Rob J. *Forecasting: methods and applications.* 3. ed. Hoboken: John Wiley & Sons, 2018.

MALKIEL, B. G. **A random walk down Wall Street : the time-tested strategy for successful investing**. New York: W.W. Norton & Company, 2019.

MAAS, Andrew L.; HANNUN, Awni Y.; NG, Andrew Y. **Rectifier Nonlinearities Improve Neural Network Acoustic Models**. In: *PROCEEDINGS OF THE 30th INTERNATIONAL CONFERENCE ON MACHINE LEARNING (ICML)*, 2013. Disponível em: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf. Acesso em: 18 fev. 2025.

MCCULLOCH, W.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biology**, v. 52, n. 1-2, p. 99–115, 1990.

MURPHY, J. J. **Technical analysis of the financial markets: a comprehensive guide to trading methods and applications**. New York: New York Institute Of Finance, 1999.

PEREA, R.; HARER, J. **Sliding window and persistence homology for time series analysis**. arXiv, 2013. Disponível em: https://arxiv.org/abs/1307.6188.

PRING, M. J. **Technical analysis explained: the successful investor's guide to spotting investment trends and turning points**. New York: Mcgraw-Hill Education, 2014.

RANKIA BRASIL. Análise técnica: princípios, teorias, figuras e indicadores. 2023. Disponível em: https://rankia.com.br/analise-tecnica/. Acesso em: 14 fev. 2025.

RYLL, L. et al. Transforming Paradigms: A Global AI in Financial Services Survey. **SSRN Electronic Journal**, 2020.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 6088, p. 533–536, out. 1986.

Resumo das operações | B3. Disponível em: <https://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/consultas/mercado-de-derivativos/resumo-das-operacoes/resumo-por-produto/>.

THAKUR, Amey; KONDE, Archit. Fundamentals of Neural Networks. Mumbai: University of Mumbai, 2021. Disponível em: https://vixra.org/pdf/2108.0130v1.pdf. Acesso em: 20 fev. 2025.

SILVA, João; MEDEIROS, Ricardo. Evidências empíricas sobre a eficácia da análise técnica no mercado financeiro brasileiro. *Revista Brasileira de Economia*, v. 71, n. 3, p. 389-412, 2017.

SOUZA, R. M.; SILVA, J. P. Aplicações de médias móveis na análise de séries temporais financeiras. *Revista Brasileira de Estatística*, Rio de Janeiro, v. 75, n. 2, p. 123-138, 2021.

STAUDEMEYER, Ralf C.; MORRIS, Eric Rothstein. **Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks**. arXiv, 2019. Disponível em: https://arxiv.org/abs/1909.09586. Acesso em: 18 set. 2024.

WILDER, J. Welles Jr. **New concepts in technical trading systems**. Greensboro/N.C.: Trend Research, 1978.

WORLD ECONOMIC FORUM; CAMBRIDGE CENTRE FOR ALTERNATIVE FINANCE. **Transforming Paradigms: A Global AI in Financial Services Survey**. 2020. Disponível em: https://www.weforum.org/publications/transforming-paradigms-a-global-ai-in-financial-services-survey/. Acesso em: 18 set. 2024.

ZHANG, Aston; LIPTON, Zachary C.; LI, Mu; SMOLA, Alexander J. **Dive Into Deep Learning**. Release 0.17.5. [S. l.]: d2l.ai, 2021. Disponível em: https://d2l.ai. Acesso em: 20 fev. 2025.

ZHANG, G. PETER. Time Series Forecasting Using a Hybrid ARIMA and Neural Network Model. **Neurocomputing**, v. 50, p. 159–175, jan. 2003.

APÊNDICE A – CÓDIGO FONTE

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_absolute_percentage_error,
mean_absolute_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, LayerNormalization,
LeakyReLU

from tensorflow.keras.optimizers import Adam, schedules

from tensorflow.keras.callbacks import EarlyStopping

import random

import os

import tensorflow as tf

import scipy.stats as stats


SEED = 42  # Escolha um número qualquer


# Fixar as seeds para garantir a reprodutibilidade

random.seed(SEED)  # Define a seed para o gerador de números aleatórios do
módulo random

np.random.seed(SEED)  # Define a seed para o gerador de números aleatórios
do NumPy
```

```
tf.random.set_seed(SEED)  # Define a seed para o gerador de números aleatórios do TensorFlow


# Garantir comportamento determinístico no TensorFlow

os.environ['TF_DETERMINISTIC_OPS'] = '1'  # Força o TensorFlow a usar operações determinísticas


data = pd.read_csv('Dados.csv')


data["Datetime"] = pd.to_datetime(data["Datetime"])

data.set_index("Datetime", inplace=True)


# Normaliza os dados utilizando o MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))  # Inicializa o escalador com o intervalo desejado (0 a 1)

scaled_data = scaler.fit_transform(data[['Close']].values)  # Aplica o escalador na coluna 'Close' dos dados


# Função para criar sequências de dados

def create_sequences(data, n_steps):

    num_samples = len(data) - n_steps  # Calcula o número de amostras possíveis

    X = np.array([data[i:i+n_steps, 0] for i in range(num_samples)])  # Cria as sequências de entrada

    y = np.array([data[i+n_steps, 0] for i in range(num_samples)])  # Cria os valores alvo correspondentes

    return X, y
```

```
n_steps = 16  # Define o número de passos (tamanho da janela) para as sequências

X, y = create_sequences(scaled_data, n_steps)  # Gera as sequências de entrada e saída


# Redimensiona X para o formato de entrada esperado pelo LSTM

X = X.reshape((X.shape[0], X.shape[1], 1))  # Adiciona uma dimensão para características (features)


# ------------------------------

# 2. Divisão em Treino, Validação e Teste

# ------------------------------

train_size = int(len(X) * 0.7)  # Define 70% dos dados para treino

val_size = int(len(X) * 0.15)   # Define 15% dos dados para validação

test_size = len(X) - (train_size + val_size)  # O restante (15%) será para teste


# Divide os dados em conjuntos de treino, validação e teste

X_train_1, X_val_1, X_test_1 = X[:train_size], X[train_size:train_size+val_size], X[train_size+val_size:]

y_train_1, y_val_1, y_test_1 = y[:train_size], y[train_size:train_size+val_size], y[train_size+val_size:]


# ------------------------------

# 3. Modelo LSTM (com 4 camadas)
```

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

model_1 = Sequential([  # Inicializa um modelo sequencial no Keras

    LSTM(units=100, return_sequences=True, input_shape=(n_steps, 1)),  # Primeira camada LSTM com 100 neurônios

    LeakyReLU(alpha=0.01),  # Função de ativação LeakyReLU para evitar o problema do neurônio morto

    LSTM(units=50, return_sequences=False),  # Segunda camada LSTM com 50 neurônios (não retorna sequências)

    LeakyReLU(alpha=0.01),  # Aplica LeakyReLU novamente

    Dense(1, activation='linear')  # Camada de saída com ativação linear para prever valores contínuos

])



# ------------------------------

# 3.1 Definição do Otimizador

# ------------------------------



# Configuração do decaimento da taxa de aprendizado (Exponential Decay)

lr_schedule = schedules.ExponentialDecay(

    initial_learning_rate=0.0001,  # Taxa de aprendizado inicial

    decay_steps=10000,  # Número de passos antes do decaimento

    decay_rate=0.9  # Fator de decaimento (a cada 10.000 steps, a LR será reduzida em 10%)

)
```

```
# Define o otimizador Adam com a taxa de aprendizado programada e limite no gradiente (clipnorm)
optimizer = Adam(learning_rate=lr_schedule, clipnorm=2)

# Compila o modelo, definindo a função de perda e métricas de avaliação
model_1.compile(
    optimizer=optimizer,  # Usa o otimizador definido acima
    loss='mean_squared_error',  # Função de perda: erro quadrático médio (MSE)
    metrics=['mae']  # Métrica adicional: erro absoluto médio (MAE)
)

# ------------------------------
# 4. Treinamento do Modelo
# ------------------------------

# Define um callback para interromper o treinamento caso a perda de validação não melhore
early_stop = EarlyStopping(
    monitor='val_loss',  # Monitora a perda do conjunto de validação
    patience=10,  # Se não houver melhora por 10 épocas, o treinamento é interrompido
    restore_best_weights=True  # Restaura os melhores pesos obtidos durante o treinamento
)
```

```
# Treina o modelo
history = model_1.fit(
    X_train_1, y_train_1,  # Dados de entrada e saída para treinamento
    validation_data=(X_val_1, y_val_1),  # Dados de validação
    epochs=100,  # Número máximo de épocas de treinamento
    batch_size=64,  # Tamanho do batch (número de amostras processadas por vez)
    callbacks=[early_stop],  # Usa o callback para parada antecipada
    shuffle=False  # Desativa a aleatorização da ordem dos dados, importante para séries temporais
)

# -------------------------------------
# 5. Predição nos Dados de Teste e Treinamento
# -------------------------------------

# Realiza a previsão nos dados de teste
y_pred_1 = model_1.predict(X_test_1)

# -------------------------------------
# 6. Função para reverter a normalização MinMaxScaler
# -------------------------------------

def inverse_transform_values_1(scaled_values):
```

```
    """Reverte a transformação do MinMaxScaler para os valores previstos ou
reais."""


    dummy = np.zeros((len(scaled_values), 1))  # Cria um array auxiliar com
o mesmo formato dos dados originais

    dummy[:, 0] = scaled_values.flatten()  # Atribui os valores escalados à
única coluna (representando 'Close')


    inv = scaler.inverse_transform(dummy)  # Aplica a transformação inversa
para recuperar os valores originais

    return inv.flatten()  # Retorna os valores transformados em um array
unidimensional


# -------------------------------------

# 7. Reversão da transformação para valores previstos e reais

# -------------------------------------


# Converte as previsões feitas pelo modelo de volta ao valor original
(desfazendo a normalização)

pred_inversed_1 = inverse_transform_values_1(y_pred_1)


# Converte os valores reais (testes) de volta à escala original

actual_inversed_1 = inverse_transform_values_1(y_test_1)


# Converte os valores reais do conjunto de treinamento para a escala
original

y_train_actual_1 = inverse_transform_values_1(y_train_1.reshape(-1, 1))
```

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

# 8. Previsões no conjunto de treinamento

# -------------------------------------


# Gera previsões no conjunto de treinamento (dados que o modelo usou para
aprender)

train_predictions_scaled_1 = model_1.predict(X_train_1)


# Converte as previsões do treinamento de volta para a escala original

train_predictions_actual_1 =
inverse_transform_values_1(train_predictions_scaled_1)


# -------------------------------------

# 9. Cálculo das Métricas de Erro

# -------------------------------------


# Calcula o MAPE (Erro Percentual Absoluto Médio) para o conjunto de
treinamento

train_mape_1 = mean_absolute_percentage_error(y_train_actual_1,
train_predictions_actual_1) * 100


# Calcula o MAPE para o conjunto de teste

test_mape_1 = mean_absolute_percentage_error(actual_inversed_1,
pred_inversed_1) * 100
```

```
# Cálculo do MAPE global (combinando dados de treino e teste)

all_actual_1 = np.concatenate([y_train_actual_1, actual_inversed_1])  #
Junta os valores reais de treino e teste

all_predicted_1 = np.concatenate([train_predictions_actual_1,
pred_inversed_1])  # Junta as previsões de treino e teste

standard_mape_1 = mean_absolute_percentage_error(all_actual_1,
all_predicted_1) * 100  # Calcula o MAPE geral


# Calcula o MAE (Erro Absoluto Médio) para o conjunto de treinamento

train_mae_1 = mean_absolute_error(y_train_actual_1,
train_predictions_actual_1)


# Calcula o MAE para o conjunto de teste

test_mae_1 = mean_absolute_error(actual_inversed_1, pred_inversed_1)


# Calcula o MAE geral (combinando treino e teste)

standard_mae_1 = mean_absolute_error(all_actual_1, all_predicted_1)


# -------------------------------------

# 10. Exibição das Métricas de Erro

# -------------------------------------


# Exibe as métricas de erro no console

print(f"MAPE Padrão (Geral): {standard_mape_1:.4f}%")  # Erro percentual
médio geral
```

```
print(f"MAPE Treino: {train_mape_1:.4f}%")  # Erro percentual médio no
treino

print(f"MAPE Teste: {test_mape_1:.4f}%")  # Erro percentual médio no teste

print(f"MAE Padrão: {standard_mae_1:.2f}")  # Erro absoluto médio geral

print(f"MAE Treino: {train_mae_1:.2f}")  # Erro absoluto médio no treino

print(f"MAE Teste: {test_mae_1:.2f}")  # Erro absoluto médio no teste


# ------------------------------

# Plotando as previsões vs os valores reais

# ------------------------------


# Define o tamanho da figura do gráfico

plt.figure(figsize=(12, 6))


# Plota a série de valores reais

plt.plot(actual_inversed_1, label='Valor Real')


# Plota a série de valores previstos

plt.plot(pred_inversed_1, label='Valor Previsto')


# Define o rótulo do eixo X como "Tempo"

plt.xlabel('Tempo')


# Define o rótulo do eixo Y como "Valor Real"
```

```
plt.ylabel('Valor Real')


# Adiciona a legenda ao gráfico

plt.legend()


# Exibe o gráfico

plt.show()


# ------------------------------

# Plotando as previsões vs os valores reais (últimos 250 pontos)

# ------------------------------


# Define o tamanho da figura do gráfico

plt.figure(figsize=(12, 6))


# Plota os últimos 250 valores reais da série

plt.plot(actual_inversed_1[len(actual_inversed_1)-250:], label='Valor
Real')


# Plota os últimos 250 valores previstos da série

plt.plot(pred_inversed_1[len(actual_inversed_1)-250:], label='Valor
Previsto')


# Define o rótulo do eixo X como "Tempo"

plt.xlabel('Tempo')
```

```
# Define o rótulo do eixo Y como "Valor Real"
plt.ylabel('Valor Real')

# Adiciona a legenda ao gráfico
plt.legend()

# Exibe o gráfico
plt.show()

data = pd.read_csv('Dados.csv')

data["Datetime"] = pd.to_datetime(data["Datetime"])
data.set_index("Datetime", inplace=True)

# Define as colunas que serão usadas como features (variáveis
independentes)
features = ['Open', 'High', 'Low', 'Close']

# Define a variável alvo (target), que será prevista pelo modelo
target = 'Close'

# ------------------------------
# 2. Divisão dos Dados em Treinamento, Validação e Teste
```

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


# Número de passos (timesteps) para a criação de sequências temporais

n_steps = 16


# Define o tamanho dos conjuntos de treino e validação

train_size = int(len(data) * 0.7)  # 70% dos dados para treino

val_size = int(len(data) * 0.15)   # 15% dos dados para validação (o
restante será para teste)


# Separa os dados de treino com base nas features selecionadas

train_data_2 = data.iloc[:train_size][features]


# Separa os dados de validação, ignorando os primeiros n_steps para evitar
sobreposição

val_data_2 = data.iloc[train_size+n_steps:train_size+val_size][features]


# Separa os dados de teste (restante dos dados após treino e validação)

test_data_2 = data.iloc[train_size+val_size:][features]


# ------------------------------

# 3. Normalização (Usar apenas os Dados de Treinamento para Ajustar o
Escalador)

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

```
# Inicializa o MinMaxScaler para normalizar os dados no intervalo de 0 a 1

scaler_2 = MinMaxScaler(feature_range=(0, 1))


# Ajusta o escalador apenas com os dados de treino para evitar vazamento de
informações

scaler_2.fit(train_data_2)


# Aplica a transformação nos conjuntos de treino, validação e teste

scaled_train_2 = scaler_2.transform(train_data_2)

scaled_val_2 = scaler_2.transform(val_data_2)

scaled_test_2 = scaler_2.transform(test_data_2)


# Converte os arrays transformados de volta para DataFrames, preservando
índices e nomes das colunas

scaled_train_df_2 = pd.DataFrame(scaled_train_2, index=train_data_2.index,
columns=features)

scaled_val_df_2 = pd.DataFrame(scaled_val_2, index=val_data_2.index,
columns=features)

scaled_test_df_2 = pd.DataFrame(scaled_test_2, index=test_data_2.index,
columns=features)


# ------------------------------

# 4. Criação de Sequências Temporais para Treinamento

# ------------------------------


# Função para criar sequências de séries temporais
```

```
def create_sequences_2(df, target_col, n_steps):

    X, y = [], []  # Listas para armazenar os inputs (X) e os valores alvo
(y)



    # Percorre os dados criando janelas deslizantes

    for i in range(n_steps, len(df)):

        X.append(df.iloc[i - n_steps:i].to_numpy())  # Seleciona os n_steps
anteriores como entrada

        y.append(df.iloc[i, df.columns.get_loc(target_col)])  # Seleciona o
valor alvo na posição atual



    return np.array(X), np.array(y)  # Converte as listas para arrays NumPy



# Define as sequências de treino, validação e teste

X_train_2, y_train_2 = create_sequences_2(scaled_train_df_2, 'Close',
n_steps)

X_val_2, y_val_2 = create_sequences_2(scaled_val_df_2, 'Close', n_steps)

X_test_2, y_test_2 = create_sequences_2(scaled_test_df_2, 'Close', n_steps)



# ☑ Redimensiona as entradas para o formato exigido pelo LSTM (amostras,
passos no tempo, número de features)

X_train_2 = X_train_2.reshape((X_train_2.shape[0], X_train_2.shape[1],
len(features)))

X_val_2 = X_val_2.reshape((X_val_2.shape[0], X_val_2.shape[1],
len(features)))

X_test_2 = X_test_2.reshape((X_test_2.shape[0], X_test_2.shape[1],
len(features)))
```

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

# 5. Modelo LSTM (Multivariado com 4 Camadas)

# ----------------------------


# Definição da arquitetura do modelo LSTM

model_2 = Sequential([

    LSTM(units=100, return_sequences=True, input_shape=(n_steps,
len(features))),  # Primeira camada LSTM com 100 neurônios

    LeakyReLU(alpha=0.01),  # Função de ativação para evitar neurônios
mortos

    LSTM(units=50, return_sequences=False),  # Segunda camada LSTM com 50
neurônios (não retorna sequência)

    LeakyReLU(alpha=0.01),  # Ativação LeakyReLU para maior estabilidade

    Dense(1, activation='linear')  # Camada de saída para prever o valor de
'Close'

])


# Definição da taxa de aprendizado com decaimento exponencial

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(

    initial_learning_rate=0.0001,  # Taxa de aprendizado inicial

    decay_steps=10000,  # Número de passos antes do decaimento

    decay_rate=0.9  # Fator de redução da taxa de aprendizado

)
```

```
# Define o otimizador Adam com a taxa de aprendizado ajustada e limitação de gradiente

optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule, clipnorm=2)


# Compila o modelo definindo a função de perda como erro quadrático médio (MSE)

model_2.compile(optimizer=optimizer, loss='mean_squared_error')


# ------------------------------

# 6. Estratégia de Parada Antecipada (Early Stopping)

# ------------------------------


early_stop_2 = EarlyStopping(

    monitor='val_loss',  # Monitora a perda no conjunto de validação

    patience=10,  # Para o treinamento se não houver melhora por 10 épocas consecutivas

    restore_best_weights=True  # Restaura os melhores pesos obtidos durante o treinamento

)


# Exibe a forma do conjunto de treino para verificação

print(X_train_2.shape)


# ------------------------------

# 7. Treinamento do Modelo

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

```
history = model_2.fit(
    X_train_2, y_train_2,  # Dados de entrada e saída para o treinamento
    validation_data=(X_val_2, y_val_2),  # Usa o conjunto de validação explicitamente
    epochs=100,  # Número máximo de épocas de treinamento
    batch_size=64,  # Tamanho do batch para otimizar o desempenho do treinamento
    callbacks=[early_stop_2],  # Utiliza a estratégia de parada antecipada
    shuffle=False  # Os dados de séries temporais não devem ser embaralhados
)

# Exibe um resumo da arquitetura do modelo treinado
model_2.summary()

# Gerar previsões para os conjuntos de teste e treino
y_pred_2 = model_2.predict(X_test_2)

# Obter o índice da coluna 'Close' na lista de features
close_index = features.index('Close')

def inverse_transform_values_2(scaled_values):
    """Função para reverter a transformação MinMax nos valores previstos ou reais."""
```

```
    dummy = np.zeros((len(scaled_values), len(features)))  # Criar um array vazio para a transformação inversa

    dummy[:, close_index] = scaled_values.flatten()  # Atribuir valores à coluna 'Close'

    inv = scaler_2.inverse_transform(dummy)  # Aplicar a transformação inversa

    return inv[:, close_index]  # Retornar apenas os valores de 'Close'


# Reverter a transformação dos valores previstos e reais

pred_inversed_2 = inverse_transform_values_2(y_pred_2)

actual_inversed_2 = inverse_transform_values_2(y_test_2)

y_train_actual_2 = inverse_transform_values_2(y_train_2.reshape(-1, 1))


# Gerar previsões para o conjunto de treino e reverter os valores

train_predictions_scaled_2 = model_2.predict(X_train_2)

train_predictions_actual_2 = inverse_transform_values_2(train_predictions_scaled_2)


# Calcular o erro percentual absoluto médio (MAPE) para treino e teste

train_mape_2 = mean_absolute_percentage_error(y_train_actual_2, train_predictions_actual_2) * 100

test_mape_2 = mean_absolute_percentage_error(actual_inversed_2, pred_inversed_2) * 100


# Calcular o MAPE padrão (considerando todos os dados: treino + teste)

all_actual_2 = np.concatenate([y_train_actual_2, actual_inversed_2])  # Combinar valores reais de treino e teste
```

```
all_predicted_2 = np.concatenate([train_predictions_actual_2,
pred_inversed_2])  # Combinar previsões de treino e teste

standard_mape_2 = mean_absolute_percentage_error(all_actual_2,
all_predicted_2) * 100  # MAPE geral


# Calcular o erro absoluto médio (MAE) para treino e teste

train_mae_2 = mean_absolute_error(y_train_actual_2,
train_predictions_actual_2)

test_mae_2 = mean_absolute_error(actual_inversed_2, pred_inversed_2)

standard_mae_2 = mean_absolute_error(all_actual_2, all_predicted_2)


# Exibir as métricas de erro

print(f"MAPE Padrão (Geral): {standard_mape_2:.4f}%")

print(f"MAPE Treino: {train_mape_2:.4f}%")

print(f"MAPE Teste: {test_mape_2:.4f}%")

print(f"MAE Padrão: {standard_mae_2:.2f}")

print(f"MAE Treino: {train_mae_2:.2f}")

print(f"MAE Teste: {test_mae_2:.2f}")


# ------------------------------

# Plotando as previsões vs os valores reais

# ------------------------------


# Define o tamanho da figura do gráfico

plt.figure(figsize=(12, 6))
```

```
# Plota a série de valores reais
plt.plot(actual_inversed_2, label='Valor Real')

# Plota a série de valores previstos
plt.plot(pred_inversed_2, label='Valor Previsto')

# Define o rótulo do eixo X como "Tempo"
plt.xlabel('Tempo')

# Define o rótulo do eixo Y como "Valor Real"
plt.ylabel('Valor Real')

# Adiciona a legenda ao gráfico
plt.legend()

# Exibe o gráfico
plt.show()

# ------------------------------
# Plotando as previsões vs os valores reais (últimos 250 pontos)
# ------------------------------

# Define o tamanho da figura do gráfico
```

```
plt.figure(figsize=(12, 6))


# Plota os últimos 250 valores reais da série

plt.plot(actual_inversed_2[len(actual_inversed_2)-250:], label='Valor
Real')


# Plota os últimos 250 valores previstos da série

plt.plot(pred_inversed_2[len(actual_inversed_2)-250:], label='Valor
Previsto')


# Define o rótulo do eixo X como "Tempo"

plt.xlabel('Tempo')


# Define o rótulo do eixo Y como "Valor Real"

plt.ylabel('Valor Real')


# Adiciona a legenda ao gráfico

plt.legend()


# Exibe o gráfico

plt.show()


data = pd.read_csv('Dados.csv')


data["Datetime"] = pd.to_datetime(data["Datetime"])
```

```
data.set_index("Datetime", inplace=True)


features = ['Open', 'High', 'Low', 'EMA', 'Close']

target = 'Close'


# ------------------------------

# 2. Train-Test-Validation Split

# ------------------------------

n_steps = 16


train_size = int(len(data) * 0.7)

val_size = int(len(data) * 0.15)


train_data_3 = data.iloc[:train_size][features]

val_data_3 = data.iloc[train_size+n_steps:train_size+val_size][features]  #
Add n_steps to avoid overlap

test_data_3 = data.iloc[train_size+val_size:][features]


# ------------------------------

# 3. Scaling (Fit only on Training Data)

# ------------------------------

scaler_3 = MinMaxScaler(feature_range=(0, 1))

scaler_3.fit(train_data_3)  # ☑ Fit only on training data to prevent
leakage
```

```
# Apply transformation

scaled_train_3 = scaler_3.transform(train_data_3)

scaled_val_3 = scaler_3.transform(val_data_3)

scaled_test_3 = scaler_3.transform(test_data_3)


# Convert back to DataFrame (preserving index & column names)

scaled_train_df_3 = pd.DataFrame(scaled_train_3, index=train_data_3.index,
columns=features)

scaled_val_df_3 = pd.DataFrame(scaled_val_3, index=val_data_3.index,
columns=features)

scaled_test_df_3 = pd.DataFrame(scaled_test_3, index=test_data_3.index,
columns=features)


# ------------------------------

# 4. Creating Time Series Sequences

# ------------------------------

def create_sequences_3(df, target_col, n_steps):

    X, y = [], []

    for i in range(n_steps, len(df)):

        X.append(df.iloc[i - n_steps:i].to_numpy())

        y.append(df.iloc[i, df.columns.get_loc(target_col)])

    return np.array(X), np.array(y)


# Define window size

X_train_3, y_train_3 = create_sequences_3(scaled_train_df_3, 'Close',
n_steps)
```

```
X_val_3, y_val_3 = create_sequences_3(scaled_val_df_3, 'Close', n_steps)

X_test_3, y_test_3 = create_sequences_3(scaled_test_df_3, 'Close', n_steps)


# ☑ Reshape input for LSTM (samples, time steps, features)

X_train_3 = X_train_3.reshape((X_train_3.shape[0], X_train_3.shape[1],
len(features)))

X_val_3 = X_val_3.reshape((X_val_3.shape[0], X_val_3.shape[1],
len(features)))

X_test_3 = X_test_3.reshape((X_test_3.shape[0], X_test_3.shape[1],
len(features)))


# ----------------------------

# 5. LSTM Model (Multivariate with 4 Layers)

model_3 = Sequential([

    LSTM(units=100, return_sequences=True, input_shape=(n_steps,
len(features))),  # First LSTM

    LeakyReLU(alpha=0.01),

    LSTM(units=50, return_sequences=False),  # Second LSTM

    LeakyReLU(alpha=0.01),

    Dense(1, activation='linear')  # Output layer (predicting 'Close')

])


lr_schedule = schedules.ExponentialDecay(

initial_learning_rate=0.0001, decay_steps=10000, decay_rate=0.9)

optimizer = Adam(learning_rate=lr_schedule, clipnorm=2)
```

```
# Compile the model
model_3.compile(optimizer=optimizer, loss='mean_squared_error')


# ------------------------------
# 6. Early Stopping Strategy
# ------------------------------
early_stop_3 = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)


print(X_train_3.shape)


# ------------------------------
# 7. Training the Model
# ------------------------------
history = model_3.fit(
    X_train_3, y_train_3,
    validation_data=(X_val_3, y_val_3),  # ☑ Using explicit validation data instead of validation_split
    epochs=100,
    batch_size=64,  # ☑ Increased batch size
```

```
    callbacks=[early_stop_3],
    shuffle=False  # ☑ Time-series data should not be shuffled
)

model_3.summary()

# Gerar previsões para os conjuntos de teste e treino
y_pred_3 = model_3.predict(X_test_3)

# Obter o índice da coluna 'Close' na lista de features
close_index = features.index('Close')

def inverse_transform_values_3(scaled_values):
    """Função para reverter a transformação MinMax nos valores previstos ou reais."""
    dummy = np.zeros((len(scaled_values), len(features)))  # Criar um array vazio para a transformação inversa
    dummy[:, close_index] = scaled_values.flatten()  # Atribuir valores à coluna 'Close'
    inv = scaler_3.inverse_transform(dummy)  # Aplicar a transformação inversa
    return inv[:, close_index]  # Retornar apenas os valores de 'Close'

# Reverter a transformação dos valores previstos e reais
pred_inversed_3 = inverse_transform_values_3(y_pred_3)
```

```
actual_inversed_3 = inverse_transform_values_3(y_test_3)

y_train_actual_3 = inverse_transform_values_3(y_train_3.reshape(-1, 1))


# Gerar previsões para o conjunto de treino e reverter os valores

train_predictions_scaled_3 = model_3.predict(X_train_3)

train_predictions_actual_3 =
inverse_transform_values_3(train_predictions_scaled_3)


# Calcular o erro percentual absoluto médio (MAPE) para treino e teste

train_mape_3 = mean_absolute_percentage_error(y_train_actual_3,
train_predictions_actual_3) * 100

test_mape_3 = mean_absolute_percentage_error(actual_inversed_3,
pred_inversed_3) * 100


# Calcular o MAPE padrão (considerando todos os dados: treino + teste)

all_actual_3 = np.concatenate([y_train_actual_3, actual_inversed_3])  #
Combinar valores reais de treino e teste

all_predicted_3 = np.concatenate([train_predictions_actual_3,
pred_inversed_3])  # Combinar previsões de treino e teste

standard_mape_3 = mean_absolute_percentage_error(all_actual_3,
all_predicted_3) * 100  # MAPE geral


# Calcular o erro absoluto médio (MAE) para treino e teste

train_mae_3 = mean_absolute_error(y_train_actual_3,
train_predictions_actual_3)

test_mae_3 = mean_absolute_error(actual_inversed_3, pred_inversed_3)

standard_mae_3 = mean_absolute_error(all_actual_3, all_predicted_3)
```

```
# Exibir as métricas de erro
print(f"MAPE Padrão (Geral): {standard_mape_3:.4f}%")
print(f"MAPE Treino: {train_mape_3:.4f}%")
print(f"MAPE Teste: {test_mape_3:.4f}%")
print(f"MAE Padrão: {standard_mae_3:.2f}")
print(f"MAE Treino: {train_mae_3:.2f}")
print(f"MAE Teste: {test_mae_3:.2f}")

# ------------------------------
# Plotando as previsões vs os valores reais
# ------------------------------

# Define o tamanho da figura do gráfico
plt.figure(figsize=(12, 6))

# Plota a série de valores reais
plt.plot(actual_inversed_3, label='Valor Real')

# Plota a série de valores previstos
plt.plot(pred_inversed_3, label='Valor Previsto')

# Define o rótulo do eixo X como "Tempo"
plt.xlabel('Tempo')
```

```
# Define o rótulo do eixo Y como "Valor Real"
plt.ylabel('Valor Real')


# Adiciona a legenda ao gráfico
plt.legend()


# Exibe o gráfico
plt.show()


# ------------------------------
# Plotando as previsões vs os valores reais (últimos 250 pontos)
# ------------------------------


# Define o tamanho da figura do gráfico
plt.figure(figsize=(12, 6))


# Plota os últimos 250 valores reais da série
plt.plot(actual_inversed_3[len(actual_inversed_3)-250:], label='Valor
Real')


# Plota os últimos 250 valores previstos da série
plt.plot(pred_inversed_3[len(actual_inversed_3)-250:], label='Valor
Previsto')
```

```
# Define o rótulo do eixo X como "Tempo"
plt.xlabel('Tempo')

# Define o rótulo do eixo Y como "Valor Real"
plt.ylabel('Valor Real')

# Adiciona a legenda ao gráfico
plt.legend()

# Exibe o gráfico
plt.show()

data = pd.read_csv('Dados.csv')

data["Datetime"] = pd.to_datetime(data["Datetime"])
data.set_index("Datetime", inplace=True)

# Define a lista de variáveis preditoras (features) usadas no modelo
features = ['Open', 'High', 'Low', 'RSI', 'FII', 'OBV',
            'BB High', 'BB Low', 'MACD', 'SMA', 'EMA', 'TRIX', 'Close']

# Define a variável alvo (target), que será prevista pelo modelo
target = 'Close'
```

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

# 2. Divisão dos Dados em Treinamento, Validação e Teste

# ------------------------------


# Define o número de passos de tempo considerados na previsão

n_steps = 16


# Define o tamanho do conjunto de treinamento como 70% dos dados

train_size = int(len(data) * 0.7)


# Define o tamanho do conjunto de validação como 15% dos dados

val_size = int(len(data) * 0.15)


# Divide os dados em treinamento, validação e teste

train_data_4 = data.iloc[:train_size][features]  # Dados de treinamento

val_data_4 = data.iloc[train_size+n_steps:train_size+val_size][features]  #
Dados de validação (evitando sobreposição com o treinamento)

test_data_4 = data.iloc[train_size+val_size:][features]  # Dados de teste


# ------------------------------

# 3. Normalização dos Dados (Usando Apenas o Conjunto de Treinamento)

# ------------------------------


# Instancia o normalizador MinMaxScaler para escalar os valores entre 0 e 1
```

```
scaler_4 = MinMaxScaler(feature_range=(0, 1))


# Ajusta o normalizador apenas nos dados de treinamento para evitar
vazamento de dados

scaler_4.fit(train_data_4)


# Aplica a transformação de normalização nos conjuntos de treinamento,
validação e teste

scaled_train_4 = scaler_4.transform(train_data_4)

scaled_val_4 = scaler_4.transform(val_data_4)

scaled_test_4 = scaler_4.transform(test_data_4)


# Converte os dados normalizados de volta para DataFrames, preservando
índices e nomes das colunas

scaled_train_df_4 = pd.DataFrame(scaled_train_4, index=train_data_4.index,
columns=features)

scaled_val_df_4 = pd.DataFrame(scaled_val_4, index=val_data_4.index,
columns=features)

scaled_test_df_4 = pd.DataFrame(scaled_test_4, index=test_data_4.index,
columns=features)


# ------------------------------

# 4. Criação das Sequências de Série Temporal

# ------------------------------


# Função para criar sequências de séries temporais com base no número de
passos (n_steps)
```

```
def create_sequences_4(df, target_col, n_steps):
    X, y = [], []
    for i in range(n_steps, len(df)):
        # Seleciona uma janela de tempo com n_steps de comprimento como entrada (X)
        X.append(df.iloc[i - n_steps:i].to_numpy())

        # Define o valor alvo correspondente como a saída (y)
        y.append(df.iloc[i, df.columns.get_loc(target_col)])
    return np.array(X), np.array(y)

# Gera as sequências de entrada (X) e saída (y) para os conjuntos de treinamento, validação e teste
X_train_4, y_train_4 = create_sequences_4(scaled_train_df_4, 'Close', n_steps)
X_val_4, y_val_4 = create_sequences_4(scaled_val_df_4, 'Close', n_steps)
X_test_4, y_test_4 = create_sequences_4(scaled_test_df_4, 'Close', n_steps)

# ☑ Ajusta o formato dos dados de entrada para o modelo LSTM (amostras, passos de tempo, variáveis preditoras)
X_train_4 = X_train_4.reshape((X_train_4.shape[0], X_train_4.shape[1], len(features)))
X_val_4 = X_val_4.reshape((X_val_4.shape[0], X_val_4.shape[1], len(features)))
X_test_4 = X_test_4.reshape((X_test_4.shape[0], X_test_4.shape[1], len(features)))
```

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

# 5. Construção do Modelo LSTM (Multivariado com 4 Camadas)

# ------------------------------


# Criação da arquitetura do modelo LSTM

model_4 = Sequential([

    # Primeira camada LSTM com 100 neurônios e retorno de sequência ativado

    LSTM(units=100, return_sequences=True, input_shape=(n_steps,
len(features))),

    LeakyReLU(alpha=0.01),  # Ativação Leaky ReLU para evitar saturação de
gradientes



    # Segunda camada LSTM com 50 neurônios e sem retorno de sequência

    LSTM(units=50, return_sequences=False),

    LeakyReLU(alpha=0.01),  # Ativação Leaky ReLU



    # Camada de saída densa com ativação linear para prever o preço de
fechamento

    Dense(1, activation='linear')

])



# Configuração do agendamento da taxa de aprendizado para redução
exponencial durante o treinamento

lr_schedule = schedules.ExponentialDecay(

    initial_learning_rate=0.0001, decay_steps=10000, decay_rate=0.9

)
```

```
# Definição do otimizador Adam com a taxa de aprendizado adaptativa e
restrição de norma para evitar explosão de gradientes

optimizer = Adam(learning_rate=lr_schedule, clipnorm=2)


# Compila o modelo definindo a função de perda como erro quadrático médio
(MSE)

model_4.compile(optimizer=optimizer, loss='mean_squared_error')


# ------------------------------

# 6. Estratégia de Early Stopping (Parada Antecipada)

# ------------------------------


# Define o critério de parada antecipada com monitoramento da perda na
validação

early_stop_4 = EarlyStopping(

    monitor='val_loss',  # Monitora a métrica de validação

    patience=10,  # Número de épocas sem melhoria antes de parar o
treinamento

    restore_best_weights=True  # Restaura os melhores pesos obtidos durante
o treinamento

)


# Exibe o formato final do conjunto de treinamento

print(X_train_4.shape)
```

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

# 7. Treinamento do Modelo

# ------------------------------


# Ajusta o modelo aos dados de treinamento

history = model_4.fit(

    X_train_4, y_train_4,  # Dados de entrada e saída para treinamento


    validation_data=(X_val_4, y_val_4),  # Utilizando dados de validação
explícitos em vez de validation_split


    epochs=100,  # Número máximo de épocas para o treinamento


    batch_size=64,  # Tamanho do lote aumentado para otimizar o desempenho
do treinamento


    callbacks=[early_stop_4],  # Utiliza a estratégia de parada antecipada
para evitar overfitting


    shuffle=False  # Os dados de séries temporais não devem ser
embaralhados

)


# Exibe o resumo da arquitetura do modelo treinado

model_4.summary()
```

```
# Gerar previsões para os conjuntos de teste e treino

y_pred_4 = model_4.predict(X_test_4)


# Obter o índice da coluna 'Close' na lista de features

close_index = features.index('Close')


def inverse_transform_values_4(scaled_values):

    """Função para reverter a transformação MinMax nos valores previstos ou reais."""

    dummy = np.zeros((len(scaled_values), len(features)))  # Criar um array vazio para a transformação inversa

    dummy[:, close_index] = scaled_values.flatten()  # Atribuir valores à coluna 'Close'

    inv = scaler_4.inverse_transform(dummy)  # Aplicar a transformação inversa

    return inv[:, close_index]  # Retornar apenas os valores de 'Close'


# Reverter a transformação dos valores previstos e reais

pred_inversed_4 = inverse_transform_values_4(y_pred_4)

actual_inversed_4 = inverse_transform_values_4(y_test_4)

y_train_actual_4 = inverse_transform_values_4(y_train_4.reshape(-1, 1))


# Gerar previsões para o conjunto de treino e reverter os valores

train_predictions_scaled_4 = model_4.predict(X_train_4)

train_predictions_actual_4 =
inverse_transform_values_4(train_predictions_scaled_4)
```

```
# Calcular o erro percentual absoluto médio (MAPE) para treino e teste

train_mape_4 = mean_absolute_percentage_error(y_train_actual_4,
train_predictions_actual_4) * 100

test_mape_4 = mean_absolute_percentage_error(actual_inversed_4,
pred_inversed_4) * 100


# Calcular o MAPE padrão (considerando todos os dados: treino + teste)

all_actual_4 = np.concatenate([y_train_actual_4, actual_inversed_4])  #
Combinar valores reais de treino e teste

all_predicted_4 = np.concatenate([train_predictions_actual_4,
pred_inversed_4])  # Combinar previsões de treino e teste

standard_mape_4 = mean_absolute_percentage_error(all_actual_4,
all_predicted_4) * 100  # MAPE geral


# Calcular o erro absoluto médio (MAE) para treino e teste

train_mae_4 = mean_absolute_error(y_train_actual_4,
train_predictions_actual_4)

test_mae_4 = mean_absolute_error(actual_inversed_4, pred_inversed_4)

standard_mae_4 = mean_absolute_error(all_actual_4, all_predicted_4)


# Exibir as métricas de erro

print(f"MAPE Padrão (Geral): {standard_mape_4:.4f}%")

print(f"MAPE Treino: {train_mape_4:.4f}%")

print(f"MAPE Teste: {test_mape_4:.4f}%")

print(f"MAE Padrão: {standard_mae_4:.2f}")
```

```
print(f"MAE Treino: {train_mae_4:.2f}")

print(f"MAE Teste: {test_mae_4:.2f}")


# ----------------------------

# Plotando as previsões vs os valores reais

# ----------------------------


# Define o tamanho da figura do gráfico

plt.figure(figsize=(12, 6))


# Plota a série de valores reais

plt.plot(actual_inversed_4, label='Valor Real')


# Plota a série de valores previstos

plt.plot(pred_inversed_4, label='Valor Previsto')


# Define o rótulo do eixo X como "Tempo"

plt.xlabel('Tempo')


# Define o rótulo do eixo Y como "Valor Real"

plt.ylabel('Valor Real')


# Adiciona a legenda ao gráfico

plt.legend()
```

```
# Exibe o gráfico

plt.show()


# ------------------------------

# Plotando as previsões vs os valores reais (últimos 250 pontos)

# ------------------------------


# Define o tamanho da figura do gráfico

plt.figure(figsize=(12, 6))


# Plota os últimos 250 valores reais da série

plt.plot(actual_inversed_4[len(actual_inversed_4)-250:], label='Valor
Real')


# Plota os últimos 250 valores previstos da série

plt.plot(pred_inversed_4[len(actual_inversed_4)-250:], label='Valor
Previsto')


# Define o rótulo do eixo X como "Tempo"

plt.xlabel('Tempo')


# Define o rótulo do eixo Y como "Valor Real"

plt.ylabel('Valor Real')
```

```
# Adiciona a legenda ao gráfico

plt.legend()



# Exibe o gráfico

plt.show()
```