Universidade Federal de Uberlândia
Instituto de Matemática e Estatística

Bacharelado em Estatística

Previsão de Vendas em Marketplace com Modelos Multi-Outputs: Estudo de Caso Olist

Gustavo Fellipe Fernandes de Paiva

Uberlândia-MG
2025

Gustavo Fellipe Fernandes de Paiva

Previsão de Vendas em Marketplace com Modelos Multi-Outputs: Estudo de Caso Olist

Trabalho de conclusão de curso apresentado à Coordenação do Curso de Bacharelado em Estatística como requisito parcial para obtenção do grau de Bacharel em Estatística.

Orientador: Prof. Dr. José Waldemar da Silva

Uberlândia-MG

2025

Universidade Federal de Uberlândia
Instituto de Matemática e Estatística

Coordenação do Curso de Bacharelado em Estatística

A banca examinadora, conforme abaixo assinado, certifica a adequação deste trabalho de conclusão de curso para obtenção do grau de Bacharel em Estatística.

Uberlândia, ____ de _____ de 20____

BANCA EXAMINADORA

_____

Prof. Dr. José Waldemar da Silva

_____

Prof. Dr. Lúcio Borges de Araújo

_____

Prof. Dr. Leandro Alves Pereira

Uberlândia-MG

2025

Resumo

O comércio eletrônico no Brasil tem apresentado um crescimento significativo nos últimos anos, impulsionado pela evolução tecnológica e pelas mudanças no comportamento do consumidor, com os consumidores optando cada vez mais por fazer compras online. Nesse contexto, o Olist, ecossistema que fornece soluções digitais para varejistas, torna-se um dos principais facilitadores para pequenos, médios e grandes comerciantes que desejam expandir seus negócios no ambiente digital. Olist fornece uma plataforma integrada que otimiza a gestão de vendas e permite a conexão com grandes mercados, aumentando assim a visibilidade e a receita do cliente.O objetivo deste trabalho é prever o volume semanal de vendas, ou seja, o número de itens vendidos por semana em diferentes produtos do catálogo da Olist, utilizando modelos de aprendizado de máquina e redes neurais. Foram comparados os seguintes algoritmos: modelo SARIMA, regressão ridge, Light Gradient Boosting Machine (LightGBM) e redes neurais recorrentes do tipo Long Short-Term Memory (LSTM).Os resultados demonstraram que a previsão de vendas em marketplaces é uma tarefa desafiadora devido à variabilidade e ao comportamento irregular das séries temporais. Nenhum modelo apresentou desempenho consistentemente superior em todos os produtos analisados. O LSTM, apesar de métricas relativamente baixas em alguns casos, mostrou limitações por gerar previsões estáticas, enquanto modelos como LightGBM e regressão ridge obtiveram melhor desempenho em determinados produtos, capturando padrões específicos de demanda. Essas evidências reforçam a complexidade do problema e a necessidade de metodologias mais avançadas para representar adequadamente a dinâmica das vendas.

Palavras-chave: aprendizado de máquina, redes neurais, previsão de vendas, comércio eletrônico, Olist.

Abstract

E-commerce in Brazil has shown significant growth in recent years, driven by technological advancements and changes in consumer behavior, with consumers increasingly choosing to shop online. In this context, Olist, an ecosystem that provides digital solutions for retailers, has become one of the main enablers for small, medium, and large merchants seeking to expand their businesses in the digital environment. Olist offers an integrated platform that optimizes sales management and enables connections with major marketplaces, thereby increasing customer visibility and revenue. The objective of this study is to forecast the weekly sales volume, that is, the number of items sold per week for different products in Olist's catalog, using machine learning models and neural networks. The following algorithms were compared: SARIMA, multiple linear regression, Light Gradient Boosting Machine (LightGBM), and Long Short-Term Memory (LSTM) neural networks. The results showed that sales forecasting in marketplaces is a challenging task due to the variability and irregular behavior of time series. No single model achieved consistently superior performance across all products analyzed. The LSTM, despite reaching relatively low error metrics in some cases, showed limitations by producing mostly static forecasts, while models such as LightGBM and ridge regression performed better for specific products, capturing particular demand patterns. These findings highlight the complexity of the problem and the need for more advanced methodologies to adequately represent the dynamics of sales.
Keywords: machine learning, neural networks, sales forecasting, e-commerce, Olist.

Sumário

1. Introdução

O crescimento acelerado do comércio eletrônico transformou dramaticamente os mercados de vendas no Brasil e no mundo. Com a digitalização das vendas, o e-commerce tornou-se uma das principais plataformas de transações comerciais, permitindo que empresas de todos os portes ofereçam produtos diretamente ao consumidor final, ampliando assim o alcance e a visibilidade de seus catálogos de produtos. Espera-se que o setor varejista brasileiro cresça de 3,2 trilhões de reais em 2023 para 3,8 trilhões de reais em 2027, com um crescimento acumulado de aproximadamente 20,2% durante este período [15]. O setor continua a ser impulsionado pelos seguintes fatores: aumento da popularidade dos dispositivos móveis e fácil acesso à Internet.

Nesse contexto, prever as vendas de produtos anunciados em catálogos de e-commerce tornou-se uma necessidade estratégica para empresas que buscam otimizar estoques, campanhas de marketing e operações logísticas. A capacidade de prever corretamente a demanda de produtos pode trazer uma série de benefícios, como previsibilidade de despesas e custos em períodos futuros, melhor gestão de estoques e melhor atendimento ao cliente. Além disso, falhas na previsão de vendas podem gerar perdas significativas, como excesso de estoque ou falta de produtos populares, afetando diretamente a lucratividade da empresa [26].

O volume de vendas em um ambiente de comércio eletrônico é afetado por diversos fatores, como sazonalidade, preço, concorrência, comportamento do consumidor e promoções. Esses fatores tornam o problema de previsão de vendas um desafio complexo que exige a utilização de métodos avançados para obtenção de previsões mais precisas. Historicamente, os métodos tradicionais de previsão de vendas, como médias móveis ou modelos lineares, têm sido amplamente utilizados. No entanto, à medida que os dados evoluem e tecnologias mais poderosas se tornam disponíveis, a aprendizagem automática tornou-se uma ferramenta poderosa para lidar com a complexidade deste cenário. [27]. O aprendizado de máquina usa algoritmos e modelos estatísticos e permite a criação de modelos preditivos que se ajustam automaticamente e aprendem com padrões em dados históricos, melhorando assim a precisão das previsões. Devido à grande quantidade de dados disponíveis nos portais de comércio eletrônico, essas tecnologias são capazes de identificar padrões que as tecnologias mais simples não conseguem detectar, tendo em conta múltiplas variáveis, como categorias de produtos, avaliações de usuários, datas de promoção e alterações de preços.

Empresas que adotam esses modelos de previsão de vendas têm conseguido não apenas otimizar seus estoques, mas também melhorar suas estratégias de marketing e ajustar preços de forma dinâmica para maximizar lucros. Empresas grandes nos setores de e-commerce e

varejo, como a Amazon [12] e Walmart [33] utilizam aprendizado de máquina em seus modelos preditivos de vendas.

Para isso, serão aplicados algoritmos de aprendizado de máquina supervisionado no desenvolvimento de modelos de regressão baseados em séries temporais com múltiplas saídas, visando prever a demanda semanal de produtos em um cenário de e-commerce. Será utilizada, especificamente, a base de dados fornecida pelo marketplace brasileiro Olist. Pretende-se treinar e avaliar modelos com os algoritmos Modelo Autorregressivo Integrado de Médias Móveis Sazonal (SARIMA), Regressão Ridge, Light Gradient Boosting Machine (LightGBM) e Redes Neurais de Memória de Longo Prazo (LSTM), comparando o desempenho de cada um na previsão de demanda. As análises, incluindo a implementação dos algoritmos, serão conduzidas com o auxílio da linguagem Python.

Este trabalho está organizado da seguinte forma: na Seção 1.1, são apresentados os objetivos da pesquisa; no Capítulo 2 descrevem-se funcionamento geral de cada algoritmo, no Capítulo 3, descrevem-se a metodologia adotada; no Capítulo 4, expõem-se os resultados obtidos; por fim, no Capítulo 5, apresentam-se as conclusões gerais do estudo e as perspectivas para trabalhos futuros.

1.1 OBJETIVOS

O objetivo deste trabalho é treinar e comparar modelos de previsão de volume de vendas com uma abordagem multi-output acessível e eficiente, utilizando dados históricos do e-commerce brasileiro Olist [25]. A solução proposta possibilitará às empresas uma compreensão mais clara do comportamento futuro de vendas de seus produtos, auxiliando na tomada de decisões mais ágeis e assertivas.

2. Fundamentação Teórica

2.1 Algoritmos

Neste trabalho, foram utilizados quatro algoritmos de regressão para prever a quantidade de produtos vendidos por dia: SARIMA, Regressão Ridge, LightGBM e redes neurais (LSTM).

O SARIMA (Modelo Autorregressivo Integrado de Médias Móveis Sazonal) foi utilizado como modelo inicial de referência para séries temporais, devido à sua capacidade de capturar padrões sazonais e tendências de maneira direta e interpretável. Ele serve como ponto de partida para a análise e permite comparar o desempenho dos modelos mais complexos.

A Regressão Ridge foi escolhida pela sua simplicidade, transparência e pelo seu amplo uso em problemas de predição quando há multicolinearidade entre as variáveis independentes, permitindo uma interpretação direta das relações lineares entre as variáveis preditoras e a variável de interesse.

O LightGBM foi utilizado por ser uma alternativa rápida e eficiente, sendo especialmente útil em cenários com alta dimensionalidade.

As redes neurais LSTM foram selecionadas por sua capacidade de capturar padrões complexos em séries temporais, incluindo dependências de longo prazo e não linearidades.

2.1.1 Modelo SARIMA

O modelo SARIMA, em inglês, *Seasonal AutoRegressive Integrated Moving Average* é uma extensão do modelo ARIMA tradicional, amplamente utilizado para previsão de séries temporais que apresentam comportamento sazonal. Na ausência de efeito sazonal, o modelo SARIMA se reduz a um ARIMA (p, d, q). Além disso, quando a série temporal é estacionária, o ARIMA se simplifica a um modelo ARMA (p, q). [31].

O SARIMA é indicado para séries temporais que exibem tendência, autocorrelação e sazonalidade, tornando-se uma ferramenta robusta para modelagem e previsão de fenômenos periódicos em diferentes domínios, como economia, clima e vendas no varejo.

Formulação do Modelo SARIMA

Um modelo ARIMA (p, d, q) é composto por três partes fundamentais:

- **AR** (*Autoregressive*): componente autorregressivo de ordem p;

- **I** (*Integrated*): número de diferenciações d aplicadas para tornar a série estacionária;
- **MA** (*Moving Average*): componente de média móvel de ordem q.

De forma geral, a formulação de um modelo ARIMA pode ser representada como [32] (Eq. 2.1):

$$\phi_p(B)(1-B)^d Y_t = \theta_q(B) Z_t \tag{2.1}$$

em que B é o operador de defasagem ($BY_t = Y_{t-1}$), $\phi_p(B)$ e $\theta_q(B)$ são polinômios de ordem p e q, respectivamente, e Z_t é um ruído branco.

Quando a série apresenta sazonalidade, utiliza-se o modelo SARIMA $(p,d,q) \times (P,D,Q)_s$, em que P, D, Q são as ordens sazonais dos componentes autorregressivo, de diferenciação e de média móvel, e s representa o período da sazonalidade. Nesse caso, a formulação geral é dada por (Eq. 2.2):

$$\Phi_P(B^s)\,\phi_p(B)(1-B)^d(1-B^s)^D Y_t = \Theta_Q(B^s)\,\theta_q(B) Z_t \tag{2.2}$$

onde:

- $\phi_p(B)$ e $\theta_q(B)$ são os polinômios não sazonais de AR e MA;
- $\Phi_P(B^s)$ e $\Theta_Q(B^s)$ representam os polinômios sazonais de AR e MA;
- $(1-B)^d$ e $(1-B^s)^D$ são os operadores de diferenciação não sazonal e sazonal;
- Z_t é o ruído branco.

Assim, o modelo SARIMA incorpora simultaneamente as estruturas de curto prazo (não sazonais) e os padrões cíclicos (sazonais), fornecendo uma modelagem mais adequada para séries temporais que exibem periodicidade [32].

Conforme apresentado em [23], o modelo SARIMA também pode ser representado pela Eq. 2.3

$$\begin{aligned}\nabla^D \Delta^d Z_t = {} & \phi_1 Z_{t-1} + \phi_2 Z_{t-2} + \cdots + \phi_p Z_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q} \\ & + \Phi_1 Z_{t-1S} + \Phi_2 Z_{t-2S} + \cdots + \Phi_P Z_{t-PS} - \Theta_1 \varepsilon_{t-1S} - \Theta_2 \varepsilon_{t-2S} - \cdots - \Theta_Q \varepsilon_{t-QS}\end{aligned} \tag{2.3}$$

em que:

- ∇^D – Ordem de integração sazonal com D diferenças;
- Δ^d – Ordem de integração não sazonal com d diferenças;

- ϕ_i – Coeficientes autorregressivos com $i = 1, 2, \ldots, p$;
- θ_i – Coeficientes de média móvel com $i = 1, 2, \ldots, q$;
- Φ_i – Coeficientes autorregressivos sazonais com $i = 1, 2, \ldots, P$;
- Θ_i – Coeficientes de média móvel sazonais com $i = 1, 2, \ldots, Q$;
- ε_t – Resíduo.

A Eq.2.3 define um modelo SARIMA$(p, d, q)(P, D, Q)[S]$, adequado para séries temporais com sazonalidade de período S.

Estimação dos Parâmetros do Modelo SARIMA

Uma vez definida a estrutura do modelo SARIMA, ou seja, as ordens não sazonais (p, d, q) e sazonais $(P, D, Q)_s$, o próximo passo consiste na estimação de seus parâmetros. Esses parâmetros correspondem aos coeficientes autorregressivos (AR), de médias móveis (MA) e, quando aplicável, aos termos sazonais, sendo ajustados de forma a representar adequadamente a dinâmica da série temporal.

A estimação é realizada, em geral, por meio do método da *Máxima Verossimilhança* (MLE, do inglês *Maximum Likelihood Estimation*), que busca encontrar os valores dos parâmetros que maximizam a probabilidade de os dados observados serem gerados pelo modelo. Alternativamente, alguns softwares implementam também o método dos *Mínimos Quadrados Condicionais* (CSS, do inglês *Conditional Sum of Squares*), que procura minimizar a soma dos quadrados dos resíduos condicionais. Em muitos casos, adota-se uma combinação entre CSS para a inicialização e MLE para o refinamento final da estimação [13, 32].

Além da estimação dos coeficientes, é fundamental selecionar a melhor configuração de ordens $(p, d, q, P, D, Q)_s$. Para isso, empregam-se critérios de informação, como o Critério de Informação de Akaike (AIC) e o Critério de Informação Bayesiano (BIC), que penalizam modelos excessivamente complexos e auxiliam na escolha do modelo que melhor equilibra ajuste e parcimônia.

Portanto, a estimação dos parâmetros no modelo SARIMA envolve tanto a definição criteriosa da estrutura do modelo quanto a determinação numérica dos coeficientes por métodos estatísticos robustos, garantindo que o modelo ajustado seja representativo e confiável para fins de previsão.

Critérios de Uso de Séries Temporais

Durante o desenvolvimento dos modelos SARIMA para previsão da demanda dos produtos, foram considerados três pressupostos fundamentais, segundo a literatura sobre séries temporais [6]:

- **Estacionariedade da série temporal:** tanto a variação quanto o padrão desta variação são constantes ao longo do tempo.

- **Resíduos como ruído branco:** os resíduos devem ser desprovidos de autocorrelação.
- **Normalidade dos resíduos:** espera-se que os resíduos sigam uma distribuição normal.

Para verificar esses pressupostos, foram aplicados os seguintes testes estatísticos:

- **Teste ADF (Augmented Dickey–Fuller):** verifica a estacionariedade da série. A hipótese nula (H_0) assume a presença de raiz unitária, ou seja, a série não é estacionária.
- **Teste de Ljung–Box:** avalia a autocorrelação dos resíduos do modelo. A hipótese nula (H_0) considera que não há autocorrelação significativa nos resíduos, ou seja, eles se comportam como ruído branco.
- **Teste de Jarque–Bera:** avalia se os resíduos seguem uma distribuição normal, com base nos coeficientes de assimetria (skewness) e curtose. A hipótese nula (H_0) assume que os resíduos possuem distribuição normal.

2.1.2 Regressão Ridge

A regressão Ridge é uma extensão da regressão linear múltipla, utilizada em problemas de predição quando há multicolinearidade entre as variáveis independentes ou quando se deseja reduzir a complexidade do modelo. Assim como a regressão múltipla tradicional, busca-se modelar a relação entre uma variável dependente (critério) e várias variáveis independentes (preditoras) [7, 17].

O modelo da regressão Ridge parte da formulação da regressão linear múltipla, mas introduz um termo de penalização sobre os coeficientes, de modo a reduzir a variância das estimativas e melhorar a capacidade preditiva do modelo.

Estimativa com Regularização Ridge

Enquanto a regressão linear múltipla estima os parâmetros pelo método dos mínimos quadrados ordinários (OLS) [20], a regressão Ridge modifica essa abordagem ao adicionar uma penalização proporcional à soma dos quadrados dos coeficientes, que atua reduzindo a magnitude dos estimadores e estabilizando a solução em relação ao problema de multicolinearidade. Assim, a formulação do estimador Ridge é dada por [22]:

$$\hat{\beta}(\lambda) = (X^T X + \lambda I)^{-1} X^T Y, \tag{2.4}$$

em que I é a matriz identidade e $\lambda \geq 0$. Para $\lambda = 0$, recupera-se a solução de mínimos quadrados ordinários, enquanto valores maiores de λ promovem maior suavização dos coeficientes. O parâmetro de regularização (λ) não é definido de forma arbitrária, mas sim ajustado como um hiperparâmetro do modelo, geralmente por meio de validação cruzada, de forma a equilibrar viés e variância e otimizar o desempenho preditivo. A validação cruzada é uma técnica estatística utilizada para avaliar o desempenho preditivo de um modelo e auxiliar na escolha de hiperparâmetros, como o valor de λ. O procedimento consiste em dividir o conjunto de

dados em k subconjuntos (ou *folds*) de tamanho aproximadamente igual. Em cada iteração, um dos subconjuntos é reservado para teste, enquanto os demais são utilizados para o treinamento do modelo. Esse processo é repetido k vezes, de forma que cada subconjunto é utilizado exatamente uma vez como conjunto de teste. Ao final, calcula-se a média das métricas de erro obtidas em cada iteração, fornecendo uma estimativa mais robusta do desempenho do modelo. Dessa forma, a validação cruzada reduz o risco de sobreajuste e permite selecionar o valor de λ que melhor equilibra o ajuste do modelo e sua capacidade de generalização.

MULTICOLINEARIDADE

Uma questão-chave é a correlação entre as variáveis independentes. Esse é um problema de dados, e não de especificação de modelo. A situação ideal para o pesquisador seria ter várias variáveis independentes que fossem altamente correlacionadas com a variável dependente, mas com baixa correlação entre si. Nesse contexto, torna-se necessário avaliar a multicolinearidade, uma vez que ela pode comprometer a precisão das estimativas e a interpretação dos coeficientes do modelo [17]. O VIF (*Variance Inflation Factor*) é uma medida que quantifica a multicolinearidade, indicando o quanto a variância de um coeficiente é inflada pela correlação entre variáveis preditoras. Um VIF elevado sugere que a variável regressora j está altamente correlacionada com as demais regressoras. Mais detalhes sobre o VIF podem ser vistos em [17] e sua expressão é apresentada na Eq. 2.5

$$\text{VIF}_j = \frac{1}{1 - R_j^2}, \tag{2.5}$$

em que R_j^2 é o coeficiente de determinação obtido no modelo ajustado para a variável explicativa j em função das demais explicativas.

Na prática, valores de VIF iguais a 1 indicam ausência de multicolinearidade, enquanto valores acima de 5 já sinalizam uma correlação moderada, e valores acima de 10 geralmente são interpretados como evidência forte de multicolinearidade. Quando isso ocorre, algumas estratégias podem ser aplicadas:

- Remoção de variáveis redundantes, mantendo apenas uma entre variáveis altamente correlacionadas;
- Combinação de variáveis correlacionadas em índices ou fatores, por exemplo via análise de componentes principais (PCA);
- Regularização por métodos como Ridge Regression, que reduzem a magnitude dos coeficientes e atenuam os efeitos da multicolinearidade;
- Coleta de mais dados, que pode reduzir a instabilidade das estimativas quando possível.

Assim, o VIF atua como um critério diagnóstico essencial para avaliar a qualidade das regressões múltiplas e guiar a escolha de variáveis e métodos adequados de modelagem.

MULTICOLINEARIDADE E REGULARIZAÇÃO

Uma das principais vantagens da regressão Ridge é sua capacidade de lidar com multicolinearidade. Diferentemente da regressão linear múltipla, onde a multicolinearidade pode inflar a variância dos estimadores, a penalização Ridge reduz esse impacto ao limitar a magnitude dos coeficientes.

Dessa forma, o modelo mantém todas as variáveis preditoras, mas com coeficientes ajustados de maneira a minimizar o efeito da correlação entre elas, resultando em um equilíbrio entre viés e variância que melhora o desempenho preditivo.

2.1.3 LIGHT GRADIENT BOOSTING MACHINE (LIGHTGBM)

ÁRVORES DE REGRESSÃO

A Árvore de Regressão é uma técnica de aprendizado de máquina que constrói um modelo semelhante a uma árvore para prever valores numéricos contínuos. Diferentemente de tarefas de classificação em que a saída é categórica, a regressão de árvore de decisão foca em estimar resultados numéricos. O algoritmo começa criando um nó raiz cujo valor predito é geralmente a média da variável dependente em todas as amostras. Em seguida, ele avalia todas as variáveis independentes e, para cada variável [34]:

- Se for numérica contínua, são testados diversos pontos de corte ao longo da distribuição da variável. Cada corte candidato é avaliado com base em métricas de qualidade, como a redução da soma dos quadrados dos resíduos (RSS), redução da variância ou erro quadrático médio (MSE).

- Se for categórica, o algoritmo considera possíveis agrupamentos das categorias e escolhe aquele que maximiza a homogeneidade dos valores da variável dependente, geralmente pela redução da variância.

A variável e o ponto de corte que proporcionam a maior melhoria na métrica escolhida são selecionados para formar a divisão do nó raiz.

Após a realização da primeira divisão, o algoritmo gera dois nós filhos, cada nó filho incorpora um intervalo específico de valores com base na condição de divisão estabelecida. O algoritmo então repete esse processo para cada nó filho, dividindo recursivamente os dados com base nas melhores variáveis independentes e nas condições definidas(profundidade máxima, número mínimo de amostras ou redução mínima de erro). Um exemplo de modelo construído a partir do algoritmo árvore de decisão, com duas variáveis regressoras, é ilustrado na Figura 2.1.

Esse processo de divisão pode ser resumido em [3]: dividir as P variáveis independentes (X_1 a X_p) em J regiões distintas (R_1 a R_j); para cada observação que se encaixar em uma determinada região, a predição é realizada pela média dos valores de resposta das observações de treino contidas nessa região.

As regiões são formadas buscando minimizar as somas dos quadrados dos resíduos, dada pela expressão apresentada na Eq. 2.6,

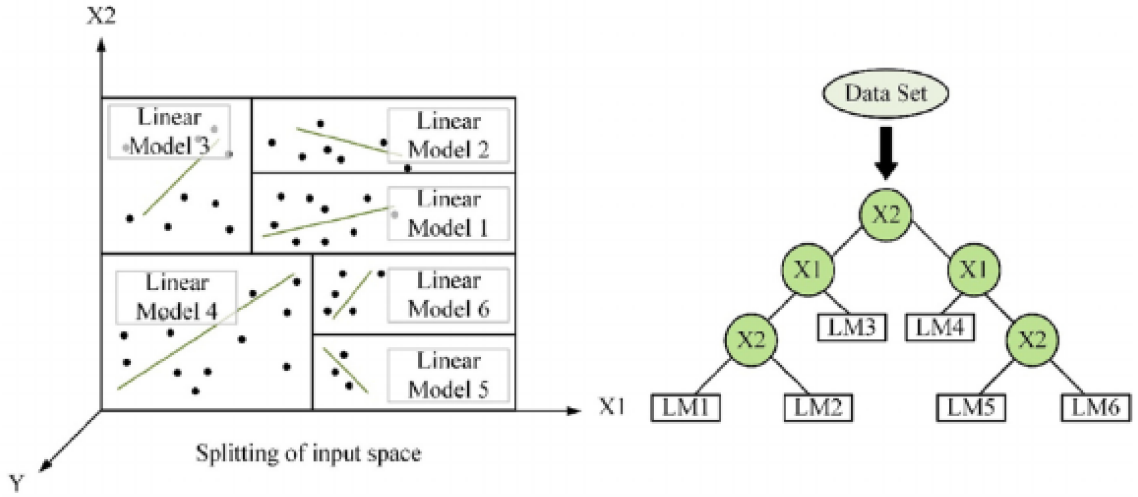

Figura 2.1: Exemplo visual de regiões baseado nas variáveis independentes.
Fonte: Imagem extraída do artigo [10].

$$SQR = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2, \tag{2.6}$$

em que y_i é o valor real e $\hat{y}_{R_j}$ é a resposta média das observações de treinamento.

Em resumo, para cada divisão candidata em cada variável, calcula-se a soma dos quadrados dos resíduos conforme a Eq. 2.6. A variável e o ponto de corte que produzem a menor soma são escolhidos para formar o nó. Esse procedimento é aplicado recursivamente em todos os nós filhos, até que um critério de parada seja atingido (profundidade máxima, número mínimo de amostras ou redução mínima de erro), garantindo que cada região seja o mais homogênea possível em relação à variável dependente.

FUNCIONAMENTO DO ALGORITMO

O Gradient Boosting Decision Tree (GBDT) é um algoritmo amplamente utilizado em aprendizado de máquina, que combina várias árvores de decisão fracas (árvores rasas, com poucos nós, que sozinhas têm desempenho limitado) para formar um modelo forte (combinação de várias árvores fracas, capaz de capturar padrões complexos). Essa metodologia baseia-se no conceito de boosting, em que, a cada etapa, uma nova árvore de decisão é treinada para prever o "erro" do modelo atual. Essa árvore é então adicionada ao modelo com sinal negativo, visando reduzir os erros do passo anterior. Esse "erro" corresponde à diferença entre as previsões do modelo e os rótulos reais [8]. O GBDT possui implementações eficientes, como o XGBoost e o pGBRT; no entanto, ainda enfrenta desafios de eficiência em conjuntos de dados grandes e de alta dimensionalidade. Nesse contexto, o LightGBM se distingue de modelos como XGBoost por utilizar algoritmos baseados em histogramas, que aceleram o processo de treinamento e

reduzem o consumo de memória, além de empregar uma estratégia de crescimento de árvores do tipo leaf-wise com restrições de profundidade [21].

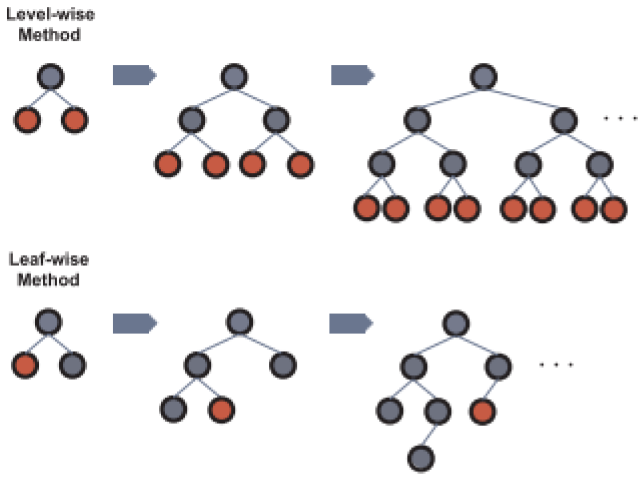

Figura 2.2: Diagrama de crescimento "nível a nível" e "por folhas".
Fonte: imagem extraída do artigo [11].

O algoritmo de histogramas funciona ao discretizar os valores contínuos das variáveis em k intervalos (ou bins) e construir um histograma com essa largura k. Com essa discretização, o algoritmo armazena apenas o intervalo (ou bin) em que cada valor cai, em vez do valor exato. Reduzindo o consumo de memória, pois não há necessidade de guardar valores precisos. Essa simplificação não compromete a precisão do modelo, pois, nas árvores de decisão, não é essencial a precisão dos pontos de divisão. De fato, essa a discretização mais grosseira das variáveis atua como uma regularização, ajudando a evitar o sobreajuste (overfitting).

Além disso, o crescimento "por folhas" (leaf-wise) o algoritmo escolhe expandir apenas a folha que mais aumenta a precisão do modelo, em vez de expandir todos os nós de um nível antes de passar para o próximo (o que ocorre no crescimento "nível a nível" ou "level-wise"). Assim, a árvore se torna mais profunda onde há maior ganho, priorizando a precisão e eficiência no uso da estrutura de nós e folhas [11].

O LightGBM tenta encontrar a melhor função $\hat{f}(x)$ possível para fazer previsões que minimize a diferença entre as previsões e os valores reais. Ele faz isso minimizando o valor esperado de uma função de perda específica $L(y, f(x))$ [11] cuja formulação é apresentada na Eq. 2.7:

$$\hat{f}(x) = \arg\min_{f} E_y, X[L(y, f(x))] \tag{2.7}$$

Para essa aproximação, o LightGBM utiliza uma série de T árvores de regressão, formando um modelo expresso como uma soma (Eq. 2.8) de T árvores individuais $f_t(X)$:

$$f_T(X) = \sum_{t=1}^{T} f_t(x) \tag{2.8}$$

Cada árvore é definida por uma função de decisão $q(x) \in \{1, 2, \ldots, N\}$, em que N é o número de folhas. Em cada folha, existe um peso ω que representa o valor estimado para as

amostras dessa folha. O modelo é então treinado de forma aditiva para melhorar o modelo atual. Esse ajuste é feito aproximando a função objetivo usando o método de Newton, o que simplifica a otimização e permite atualizações rápidas. No passo t essa função é dada por:

$$\Gamma_t \approx \sum_{j=1}^{N} L\left(y_i, F_{(t-1)}(x_i) + f_t(x_i)\right) \tag{2.9}$$

em que: $F_{(t-1)}$ é a previsão acumulada até a árvore $t-1$ e $f_t(x_i)$ é a nova árvore adicionada no passo t. Para facilitar os cálculos durante o treinamento, o LightGBM utiliza os gradientes de primeira e segunda ordens g_i e h_i da função de perda, que representam, respectivamente, a inclinação e a curvatura dessa função. Com base nesses gradientes, a função objetivo que o algoritmo busca minimizar em cada passo de treinamento pode ser expressa conforme apresentado na Eq. 2.10

$$\Gamma_t = \sum_{j=1}^{J} \left((\sum_{i \in I_j} g_i)\omega_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)\omega_j^2 \right), \tag{2.10}$$

em que J é o número total de regiões (ou folhas) na árvore de decisão, R_j representa o conjunto de amostras que caem na região j e ω_j é o peso atribuído à região j.

Em termos da estrutura da árvore $q(x)$, os pesos ótimos das folhas ω_j^* e os valores extremos de Γ_T^* são determinados, respectivamente, pelas Eqs. 2.11 e 2.12

$$\omega_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{2.11}$$

$$\Gamma_T^* = -\frac{1}{2} \sum_{j=1}^{J} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} \tag{2.12}$$

Na Eq. 2.12, Γ é a função de peso que mede a qualidade da estrutura da árvore $q(x)$.

A função objetivo é finalmente obtida calculando o ganho G da divisão

$$G = \frac{1}{2} \left(\frac{\left(\sum_{i \in I_l} g_i\right)^2}{\sum_{i \in I_l} h_i + \lambda} + \frac{(\sum_{i \in I_r} g_i)^2}{\sum_{i \in I_r} h_i + \lambda} + \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right), \tag{2.13}$$

em que I_l e I_r são as amostras do ramo esquerdo e direito, respectivamente. O ganho G da Eq. 2.13 é calculado para cada possível ponto de divisão e o ponto que maximiza G é escolhido para a próxima divisão da árvore.

2.1.4 REDES NEURAIS ARTIFICIAIS

Redes Neurais Artificiais (RNAs) são algoritmos computacionais que buscam simular a estrutura neural humana, que adquire conhecimento por experiências prévias. Essas estruturas neurais são compostas por unidades de processamento, interconectadas por canais de comunicação associados a pesos sinápticos A inteligência da estrutura se deve à iteração entre as unidades de entrada e seus respectivos pesos. Esses algoritmos possuem diversos modelos para a regra de treinamento, a partir dos quais a rede neural aprende e aprimora seu desempenho[18]. A estrutura básica de uma RNA possui uma camada de entrada com pontos iniciais, camadas ocultas com neurônios e uma camada de saída, por onde aparecem os resultados, ilustrado na Figura 2.3.

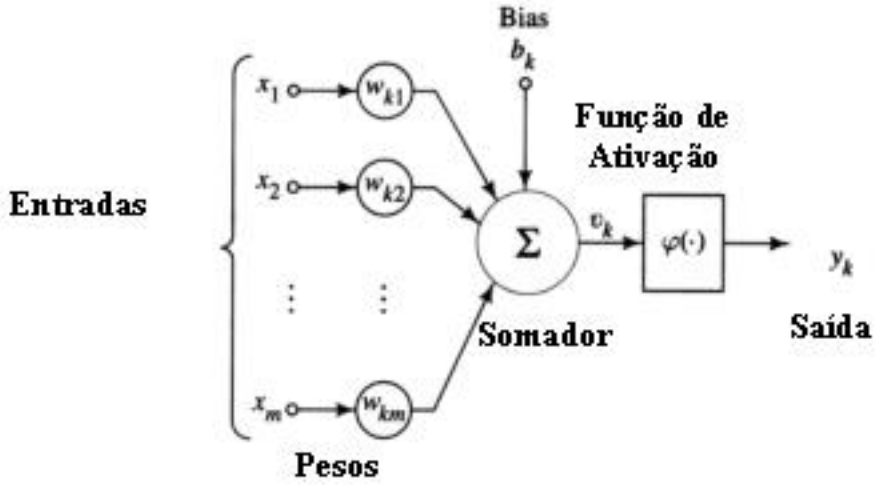

Figura 2.3: Estrutura de uma rede neural artificial (RNA).
Fonte: Imagem extraída de [16].

Uma Rede Neural Recorrente (RNN) é um tipo de rede neural artificial projetada para reconhecer padrões em sequências de dados, amplamente utilizada em dados de séries numéricas que emanam de sensores, bolsas de valores e agências governamentais [30]. As redes neurais recorrentes (Figura 2.4) apresentam ciclos entre seus neurônios. Em outras palavras, um neurônio pode estabelecer conexões com unidades de camadas anteriores, seguintes ou até com unidades da mesma camada. Dessa forma, o fluxo de informação não ocorre em um único sentido, e a saída da rede passa a depender não apenas da entrada atual, mas também das entradas anteriores. Esse mecanismo resulta, na prática, em uma memória de curto prazo na rede [24].

TREINAMENTO E ESTIMAÇÃO DOS PARÂMETROS EM RNAS

O funcionamento de uma rede neural artificial depende fortemente dos valores atribuídos aos seus pesos, que determinam a intensidade das conexões entre os neurônios. Esses pesos não são definidos de forma arbitrária, mas estimados por meio de algoritmos de otimização, cujo objetivo é minimizar uma função de perda que mede o erro entre as previsões da rede e os valores reais da série temporal.

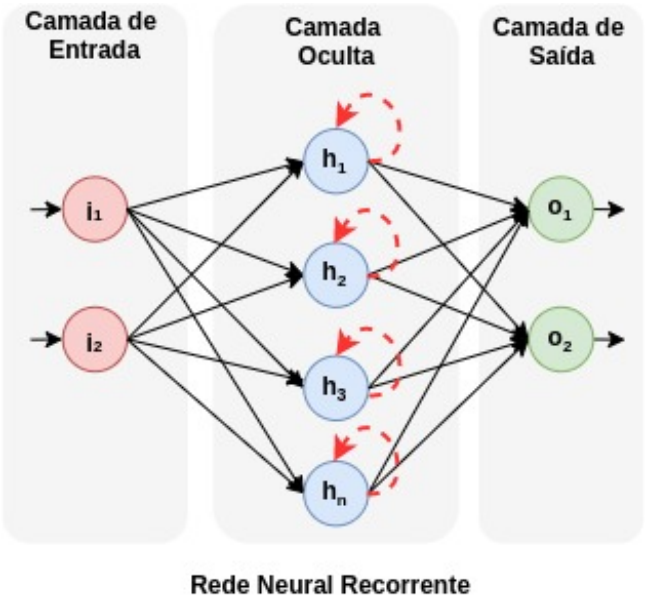

Figura 2.4: Estrutura de uma rede neural recorrente (RNN).
Fonte: Imagem extraída de [5].

O processo de treinamento é realizado iterativamente com base no algoritmo de retropropagação do erro (*backpropagation*), aliado a métodos de otimização como o Gradiente Descendente Estocástico (*Stochastic Gradient Descent*, SGD) ou suas variações (Adam, RMSprop, entre outros). A cada iteração, os gradientes da função de perda em relação aos pesos são calculados e utilizados para atualizar os parâmetros da rede, reduzindo progressivamente o erro de previsão [19].

A escolha da função de perda depende do tipo de problema. Para tarefas de regressão, como previsão de vendas, são comumente utilizadas métricas como o Erro Quadrático Médio (MSE) ou o Erro Absoluto Médio (MAE). Além disso, métricas adicionais, como RMSE, MAPE ou sMAPE, podem ser avaliadas no conjunto de validação para auxiliar na seleção do modelo mais adequado.

No caso específico das LSTMs, descritas pelas Equações 2.14, o processo de treinamento segue a mesma lógica, mas é necessário lidar com a retropropagação através do tempo (*Backpropagation Through Time*, BPTT), técnica que adapta o cálculo dos gradientes para capturar as dependências temporais presentes nas sequências. A principal diferença em relação à retropropagação convencional é que, em uma RNN ou LSTM, o erro em um determinado passo temporal depende não apenas da saída atual, mas também das entradas e estados anteriores. O BPTT calcula os gradientes levando em consideração essas dependências ao longo de toda a sequência, permitindo atualizar os pesos de forma a capturar relações temporais e padrões sequenciais nos dados. Essa abordagem é fundamental para que redes recorrentes consigam aprender dependências de curto e longo prazo em séries temporais, algo que redes feedforward tradicionais não conseguem modelar.

REDES NEURAIS DE LONGO PRAZO LSTM

Uma Rede Neural de Longo prazo (LSTM) é um tipo especial de Rede Neural Recorrente (RNN) junto com um algoritmo de aprendizado baseado em gradiente apropriado, projetado para superar problemas de retropropagação de erro [19]. Sua principal vantagem em relação às redes neurais recorrentes (RNNs) convencionais é sua capacidade de manter um estado de célula, c_t, ao longo da sequência de entrada (como uma série temporal), filtrando quais informações devem ser armazenadas ou esquecidas. Esse processo é feito por meio de três portas: a porta de entrada i_t, a porta de esquecimento f_t e a porta de saída o_t, que agem como filtros para gerenciar o fluxo de informação. Cada porta produz uma variável de estado no tempo t, junto com a célula h_t [28]. As equações abaixo descrevem esse mecanismo de forma sequencial: primeiro, calculam-se os valores das portas de entrada e esquecimento (i_t e f_t), em seguida atualiza-se o estado da célula (c_t) combinando informação nova e passada, depois define-se a porta de saída (o_t) e, por fim, obtém-se a saída oculta h_t, que é transmitida para o próximo passo temporal.

$$i_t = \sigma(W^i x_t + R^i h_{t-1} + U^i \circ c_{t-1} + b^i), \tag{2.14}$$
$$f_t = \sigma(W^f x_t + R^f h_{t-1} + U^f \circ c_{t-1} + b^f), \tag{2.15}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W^c x_t + R^c h_{t-1} + b^c), \tag{2.16}$$
$$o_t = \sigma(W^o x_t + R^o h_{t-1} + U^o \circ c_t + b^o), \tag{2.17}$$
$$h_t = o_t \circ \tanh(c_t), \tag{2.18}$$

em que:

1. σ: Função sigmoide, uma função de ativação do neurônio, que limita o valor de saída para a próxima camada da rede, entre 0 e 1, ajudando a controlar a influência de cada componente.

2. $(W, R, \text{e}\, U)$: Matrizes de pesos associadas à entrada atual x_t, saída anterior h_{t-1}, e ao estado da célula c_t, respectivamente.

3. tanh: Função de ativação tangente hiperbólica, que normaliza os valores entre -1 e 1, útil para as representações internas do estado.

4. $\circ$: Produto elemento a elemento, aplicando uma operação individualmente entre os elementos de dois vetores.

2.2 MÉTRICAS DE AVALIAÇÃO

Modelos de previsão, como aqueles utilizados para análise de séries temporais, devem ser avaliados quanto a adequação e para isso utiliza-se métrica de avaliação/validação que fornecem

informações sobre quão um modelo se ajusta aos dados e quão é adequado para fazer previsões. As métricas de avaliação servem para fornecer uma avaliação objetiva do desempenho do modelo, possibilitando a comparação entre diferentes modelos.

Uma métrica típica para modelos de regressão é a Raiz do Erro Quadrático Médio, em inglês, *Root Mean Square Error* (RMSE). A expressão do RMSE é apresentada na Eq. 2.19 e seu valor é obtido a partir do cálculo da raiz quadrada da média dos quadrados das diferenças entre os valores reais e os previstos pelo modelo. A RMSE dá uma ideia da quantidade de erros gerados pelo modelo em suas previsões, atribuindo um peso maior para grandes erros [14].

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.19}$$

Em alguns contextos, como quando existem outliers na base de dados, pode ser preferível utilizar O Erro Médio Absoluto (Eq. 2.2), em inglês, *Mean Absolute Error* (MAE). A MAE é a média das diferenças absolutas entre os valores reais e os previstos pelo modelo.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

O Erro Percentual Absoluto Médio Simétrico, em inglês, *symmetric Mean Absolute Percentage Error* (sMAPE) é uma métrica para avaliar a acurácia de previsões, oferecendo vantagens como independência de escala e interpretabilidade [6]. O sMAPE (Eq. 2.20) permite ao analista calcular o erro relativo de forma simétrica, normalizando a diferença entre os valores reais e previstos pelo somatório da média do erro absoluto com a média dos valores observados e previstos para cada período. Essa abordagem proporciona maior estabilidade em séries com valores próximos de zero e facilita a comunicação dos resultados para diferentes contextos.

$$\text{sMAPE} = \frac{1}{n}\sum_{t=1}^{n}\frac{|A_t - F_t|}{(|A_t| + |F_t|)/2} \times 100 \tag{2.20}$$

em que: A_t e F_t são os valores reais e previstos no período t, e n é o número de observações.

3. METODOLOGIA

3.1 DADOS

Os dados utilizados neste trabalho foram disponibilizados pela Olist, abrangendo informações de 97.559 pedidos realizados entre setembro de 2016 e setembro de 2018. Esses dados estão estruturados em um formato de banco de dados relacional, conforme ilustrado na Figura 3.1, contendo múltiplas tabelas dimensionais que fornecem informações detalhadas sobre o status dos pedidos, preços, métodos de pagamento, custos de frete, atributos dos produtos e avaliações dos clientes [1].

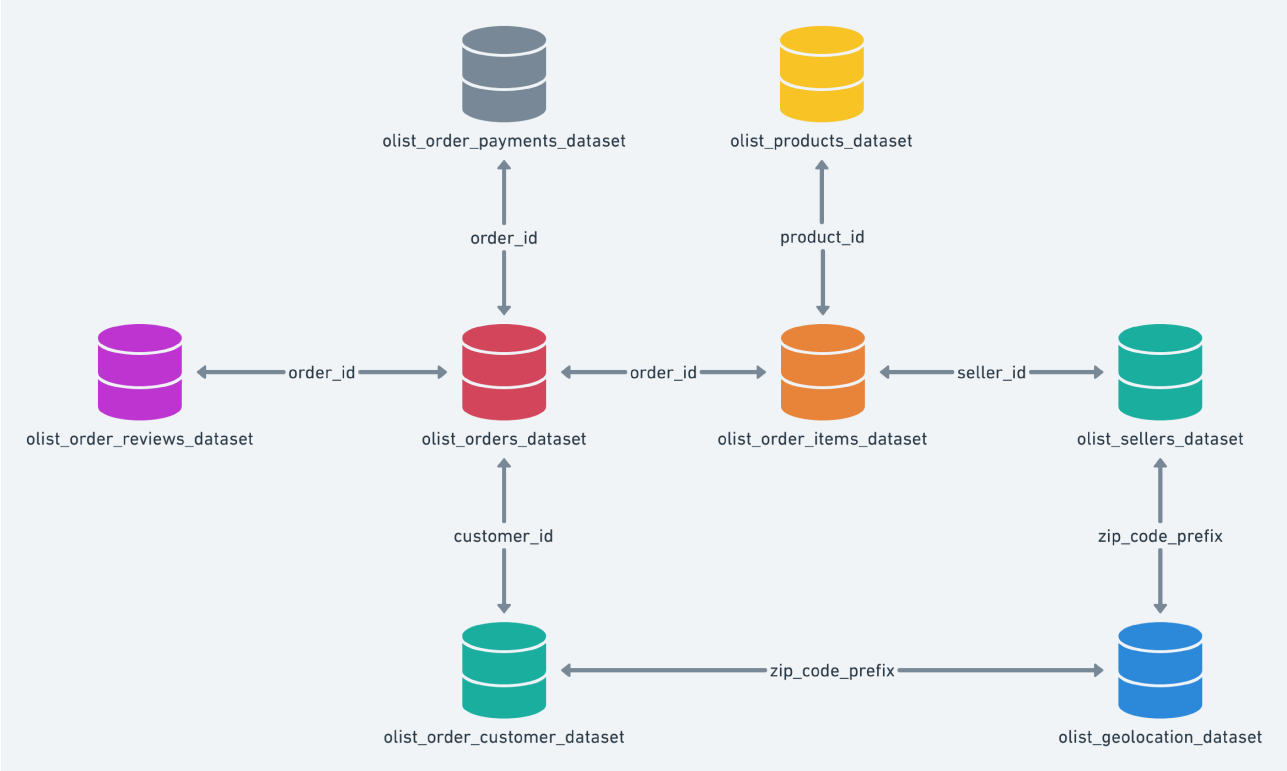

Figura 3.1: Esquema de relações das tabelas usadas.

Essas diversas fontes de dados são fundamentais para o processo de engenharia de variáveis, que consiste na criação e seleção de variáveis de entrada, também conhecidas como features, com o objetivo de melhorar o desempenho e a precisão dos modelos de aprendizado de máquina [4].

As variáveis selecionadas e criadas a partir dos dados que foram usadas para os modelos são: categoria do produto, nota média de avaliação, preço médio do produto, quantidade de vendedores que vendem o produto, tempo médio de entrega e a quantidade de unidades vendidas do produto, que é a variável resposta.

Para fins de ilustração, este trabalho considerou apenas os 10 produtos mais vendidos (com maior volume de vendas). Contudo, a abordagem proposta pode ser aplicada ao conjunto completo de produtos.

A divisão entre os conjuntos de treino e teste foi realizada de forma temporal, de modo a refletir um cenário mais próximo da realidade preditiva. Para o conjunto de teste, foram separados os últimos três meses disponíveis no dataset, correspondendo ao período entre a semana 26 e a semana 36 do ano de 2018. O restante dos dados foi utilizado para o treino dos modelos.

3.2 Variáveis Criadas

A construção de algumas variáveis envolveu cálculos específicos:

- **Nota média de avaliação:** corresponde à média das notas atribuídas pelos clientes no período considerado para o treino.
- **Preço médio do produto:** corresponde à média dos preços praticados para cada produto durante o período de treino.
- **Preço médio do produto:** corresponde à média dos preços praticados para cada produto durante o período de treino.
- **Tempo médio de entrega:** foi calculado como a média da diferença, em dias, entre a data de faturamento do pedido e a data de registro da entrega, também considerando apenas o conjunto de treino.
- **Feriados Nacionais:** foram obtidos por meio do pacote holidays do Python [29].

Além dessas, foi criada a variável que corresponde ao número de dias entre a última venda registrada e a data da transação. Também foram usadas variáveis de fontes externas referentes à sazonalidade, como: dias da semana, mês e ano da venda, além de variáveis indicadoras de feriados.

3.3 Análise Exploratória

Antes da etapa de modelagem, foi conduzida uma análise exploratória dos dados com o objetivo de compreender o comportamento das variáveis e suas potenciais relações com a variável resposta. Esse processo envolveu a aplicação de técnicas estatísticas e de visualização para:

- examinar a distribuição das vendas ao longo do tempo;
- avaliar presença de padrões sazonais;
- investigar a relação entre variáveis explicativas criadas, em relação ao volume de vendas;

  /identificar possíveis outliers ou inconsistências nos dados que pudessem impactar a modelagem;

Para auxiliar nessa etapa, foram construídas representações gráficas e estatísticas descritivas, incluindo gráficos de barras, boxplots e séries temporais, além de medidas resumo como média, mediana e variância.

Essa análise preliminar teve caráter exclusivamente descritivo e serviu como base para a escolha das variáveis incluídas na modelagem.

3.4 Implementação dos Algoritmos

Nesta seção são descritas as configurações de modelagem implementadas para a previsão do volume de vendas. Foram considerados os algoritmos: SARIMA, Regressão Ridge, LightGBM e redes neurais do tipo LSTM.

A avaliação e comparação do desempenho dos modelos foram realizadas por meio das métricas Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) e Symmetric Mean Absolute Percentage Error (sMAPE), apresentadas, respectivamente, nas Eqs. 2.19, 2.2 e 2.20.

3.4.1 Modelo SARIMA

Diferentemente dos demais modelos avaliados, o SARIMA foi ajustado sem a inclusão de variáveis exógenas, considerando apenas a própria série histórica de vendas como fonte de informação para a previsão.

Para cada produto, os parâmetros do modelo p, d, q, P, D e Q foram definidos automaticamente por meio da função do python auto arima [2], que realiza uma busca orientada no espaço de combinações possíveis, utilizando como critério de seleção o Akaike Information Criterion (AIC). Essa abordagem permitiu encontrar combinações de parâmetros adequadas à dinâmica individual de cada série, equilibrando ajuste e parcimônia. A utilização da função auto arima requer a indicação de valores máximos para p, d, q, P, D e Q. Neste trabalho estes valores máximos foram definidos conforme apresentados na Tabela 3.1.

Tabela 3.1: Valores máximos testados para os hiperparâmetros do `auto_arima`

Hiperparâmetro	Valor máximo testado
p (AR)	3
d (diferenças não sazonais)	3
q (MA)	3
P (AR sazonal)	2
D (diferenças sazonais)	3
Q (MA sazonal)	2

Após o ajuste de cada modelo, os resíduos foram analisados para verificar a adequação dos pressupostos do SARIMA. Três testes estatísticos foram empregados:

- **Teste de Ljung-Box**: utilizado para verificar a ausência de autocorrelação nos resíduos.
- **Teste de normalidade (Jarque-Bera)**: empregado para avaliar se os resíduos apresentavam distribuição aproximadamente normal.
- **Teste de raiz unitária (ADF – Augmented Dickey-Fuller)**: aplicado para verificar a estacionariedade da série original e dos resíduos.

3.4.2 Regressão Ridge

Inicialmente, foram calculados os Fatores de Inflação da Variância (VIFs) sobre as variáveis independentes, de modo a verificar a magnitude da multicolinearidade.

No presente trabalho, a matriz de preditores X, foi construída a partir das variáveis contínuas e categóricas, sendo estas últimas transformadas em variáveis indicadoras (dummies). Assim, cada produto possui uma matriz X própria, composta pelas observações de vendas no período de treino e pelas colunas correspondentes às variáveis explicativas. Após a aplicação do one-hot encoding, a matriz X totalizou 85 colunas (incluindo tanto variáveis contínuas quanto dummies).

O parâmetro de penalização λ foi ajustado via Random Search, um método de busca aleatória de hiperparâmetros que testa diferentes valores obtidos aleatoriamente a partir de um intervalo previamente fixado. Cada valor proposto é avaliado a partir de uma validação cruzada. Neste trabalho foram sorteados 100 valores no intervalo [0,001; 100]. Para cada produto, o valor ótimo de λ foi selecionado com base no menor erro quadrático médio (RMSE) obtido no conjunto de validação. Dessa forma, foi possível obter uma solução regularizada que preserva a interpretabilidade dos coeficientes e garante maior robustez preditiva.

A configuração final das variáveis preditoras é apresentada na tabela 3.2.

Tabela 3.2: Descrição das variáveis utilizadas no modelo de Regressão Ridge.

Variável	**Descrição**
Número de semanas desde o início da série temporal(X_0)	Contador de semanas (tempo)
Nota média do produto(X_1)	Avaliação média do produto
Preço médio do produto na semana(X_2)	Preço médio semanal
Quantidade de vendedores ativos(X_3)	Número de vendedores ofertando o produto
Tempo médio de entrega (em dias)(X_4)	Prazo médio de entrega
Black Friday(X_5)	Indicador de semana contendo Black Friday
Confraternização Universal(X_6)	Indicador do feriado 01 de janeiro
Dia do Trabalhador(X_7)	Indicador do feriado 01 de maio
Finados(X_8)	Indicador do feriado 02 de novembro
Independência do Brasil(X_9)	Indicador do feriado 07 de setembro
Natal(X_{10})	Indicador do feriado 25 de dezembro
Nossa Senhora Aparecida(X_{11})	Indicador do feriado 12 de outubro
Proclamação da República(X_{12})	Indicador do feriado 15 de novembro
Sexta-feira Santa(X_{13})	Indicador do feriado móvel
Tiradentes(X_{14})	Indicador do feriado 21 de abril
Semana do ano($X_{15} \cdots X_{66}$)	Dummies para cada semana do ano
Ano de referência($X_{67} \cdots X_{69}$)	Dummies para ano
Mês do ano($X_{70} \cdots X_{81}$)	Dummies para meses do ano
Trimestre do ano($X_{82} \cdots X_{85}$)	Dummies para trimestres
Produto específico(X_{86})	Dummy para produto específico
Categoria de produto(X_{87})	Dummy para categoria de produto

3.4.3 Light Gradient Boosting Machine (LightGBM)

Para cada produto, foi ajustado individualmente um modelo LightGBM (LGBM), e a busca pelos melhores hiperparâmetros foi realizada, assim como para a regressão ridge, por meio de uma técnica de Random Search, avaliando o desempenho do modelo em cada conjunto utilizando o RMSE como critério de seleção. O universo de parâmetros explorados foi definido conforme a Tabela 3.3.

Os principais hiperparâmetros otimizados e suas funções são:

- **Número de folhas**: define o número máximo de folhas em cada árvore, controlando a complexidade do modelo e sua capacidade de capturar padrões nos dados.
- **Profundidade máxima**: limita a profundidade das árvores, prevenindo overfitting ao evitar árvores excessivamente complexas.
- **Taxa de aprendizado**: regula o quanto cada árvore contribui para o modelo final, influenciando a velocidade de aprendizado e a robustez do ajuste.

- **Número de estimadores**: corresponde à quantidade máxima de árvores construídas, impactando o equilíbrio entre viés e variância.

- **Amostras mínimas por folha**: estabelece o número mínimo de observações necessárias para formar uma folha, ajudando a controlar o overfitting e garantindo que as divisões sejam estatisticamente relevantes.

Tabela 3.3: Valores dos hiperparâmetros testados nos modelos LightGBM.

Hiperparâmetro	**Valores Testados**
Número de folhas	Valores inteiros de 5 a 30
Profundidade máxima	Valores inteiros de 3 a 9
Taxa de aprendizado	Uniforme entre 0,01 e 0,31
Número de estimadores	Valores inteiros de 50 a 199
Amostras mínimas por folha	Valores inteiros de 5 a 29

Ao contrário da regressão Ridge, o modelo LGBM não requer dummificação das variáveis categóricas em variáveis indicadoras (one-hot encoding). Modelos baseados em árvores podem lidar diretamente com variáveis categóricas, dividindo os nós da árvore com base em valores discretos. Árvores conseguem identificar automaticamente as categorias mais relevantes para a previsão, focando em divisões que maximizam a redução do erro, sem a necessidade de codificação manual.

Dessa forma, as variáveis contínuas foram utilizadas diretamente e as categóricas foram passadas na forma original para o modelo, codificadas como números, permitindo que o LGBM explorasse a relação entre os atributos e o volume de vendas de forma mais eficiente e interpretável.

3.4.4 Redes Neurais de Longo prazo LSTM

Para cada produto, foi ajustado individualmente um modelo de rede neural recorrente LSTM, projetado para capturar dependências temporais de longo prazo nas séries de vendas semanais. A modelagem individual permite que o comportamento específico de cada produto, incluindo padrões sazonais e variações de demanda, seja adequadamente representado.

As variáveis contínuas foram normalizadas utilizando a função python *MinMaxScaler* [9], garantindo que todos os atributos tivessem valores entre 0 e 1, o que favorece a convergência da rede neural durante o treinamento.

As variáveis categóricas temporais foram transformadas em variáveis indicadoras (dummies) para que fossem incorporadas como entradas numéricas na rede. Esse processo permitiu que a LSTM explorasse padrões sazonais sem perder informação sobre a ordem ou frequência de cada categoria. As variáveis de entrada utilizadas nos modelos LSTM correspondem às mesmas apresentadas na Tabela 3.2, originalmente definidas para a regressão Ridge.

Os dados foram organizados em sequências temporais de comprimento definido, de modo que cada entrada da rede representasse uma janela de observações passadas, e a saída corresponderia à previsão do próximo período.

A definição dos hiperparâmetros foi realizada por meio de uma busca aleatória utilizando a função python *Random Search*, sobre um espaço de combinações previamente definido. Essa abordagem possibilitou adaptar a arquitetura do modelo às características de cada série temporal. Deste universo, foi arbitrado o total de 20 combinações a serem buscadas pela função *Random Search*.

Os principais hiperparâmetros ajustados e suas respectivas funções são:

- **Número de unidades ocultas**: define a capacidade de memória da rede e sua habilidade de capturar padrões complexos nas séries.
- **Número de camadas**: quantidade de camadas LSTM empilhadas, que afeta a profundidade da rede e a representação de dependências de longo prazo.
- **Taxa de abandono**: probabilidade de desativar neurônios durante o treinamento, com o objetivo de reduzir sobreajuste (*overfitting*).
- **Taxa de aprendizado**: define a taxa de aprendizado do modelo, controlando o tamanho dos passos durante a atualização dos pesos na otimização.

O espaço de busca para os hiperparâmetros da LSTM foi definido conforme a Tabela 3.4.

Tabela 3.4: Universo de hiperparâmetros testados para os modelos LSTM.

Hiperparâmetro	**Valores testados**
N° de unidades ocultas	16; 32; 64; 128
N° de camadas	1; 2; 3
Taxa de abandono	0,0; 0,2; 0,3; 0,5
Taxa de aprendizado	0,0001; 0,0005; 0,001; 0,005

Os modelos foram treinados por até 1000 épocas, com critério de parada configurado para 30 épocas sem melhoria no conjunto de validação.

4. RESULTADOS

Ao longo deste estudo, foram empregadas técnicas avançadas de análise de dados, como o aprendizado de máquina, para identificar padrões e tendências que possam auxiliar na predição da demanda semanal. Este capítulo se propõe a apresentar de forma detalhada os resultados obtidos, incluindo a análise das variáveis e a avaliação do desempenho do modelo.

4.1 ANÁLISE EXPLORATÓRIA

Nesta seção, são apresentados os principais aspectos estatísticos e comportamentais das variáveis presentes no conjunto de dados, com o intuito de compreender melhor suas possíveis relações com a variável resposta. A análise exploratória é fundamental para:

- Identificar tendências, sazonalidades e padrões de comportamento das vendas ao longo do tempo;
- Detectar possíveis outliers ou inconsistências nos dados que possam afetar a modelagem;
- Informar a escolha de variáveis e a criação de novas features, como variáveis temporais ou de avaliação logística;
- Fornecer insights iniciais sobre relações entre atributos, permitindo decisões mais fundamentadas na construção do modelo.

As análises a seguir foram essenciais para embasar o desenvolvimento do modelo preditivo. Os identificadores dos produtos, originalmente anonimizados em formato hash , foram mapeados para valores inteiros sequenciais exclusivamente para fins de visualização nos gráficos. Essa transformação não impacta os resultados nem o comportamento dos modelos de previsão. O gráfico de ranking (Figura 4.1) serve para identificar quais itens concentram o maior volume de vendas, permitindo priorizar análises detalhadas nesses produtos.

A Figura 4.2 mostra a evolução semanal do volume de vendas, ajudando a visualizar padrões sazonais e mudanças de comportamento do consumidor, essenciais para previsão de demanda e planejamento de estoque.

As variáveis temporais, como trimestre, mês e semana do ano, podem influenciar significativamente o comportamento de compra dos consumidores. Esta análise busca identificar padrões sazonais de consumo associados a esses agrupamentos temporais. As Figuras 4.3, 4.4 e 4.5 apresentam a média de vendas agrupadas por trimestre, mês e semana do ano, respectivamente.

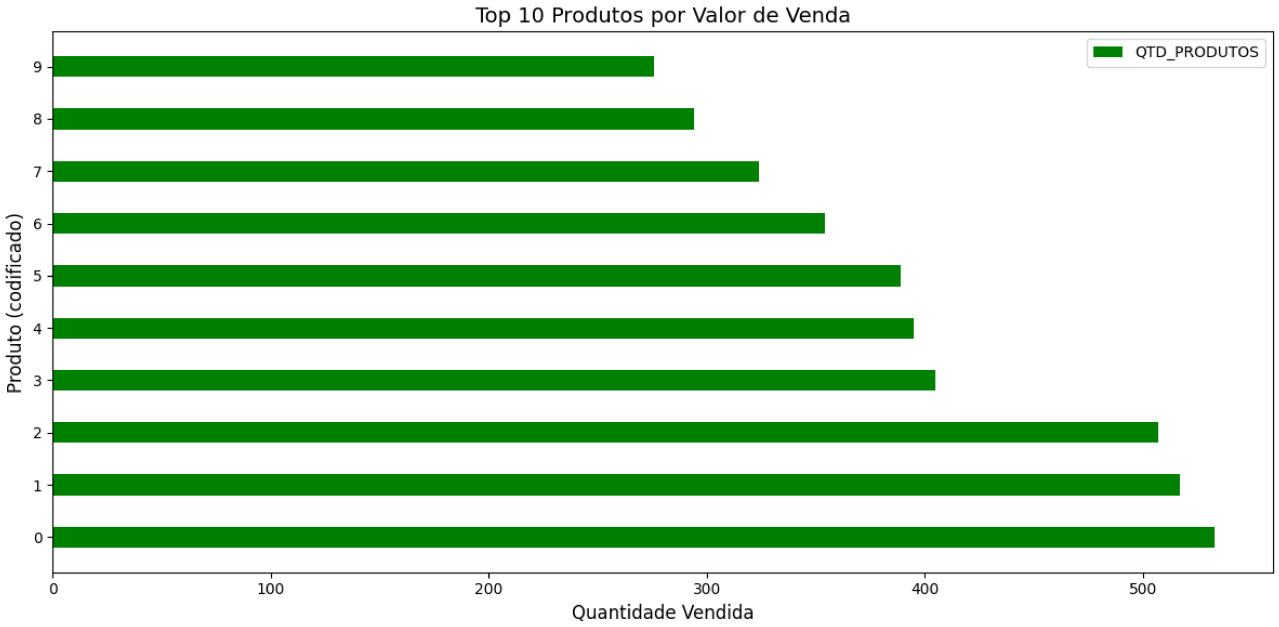

Figura 4.1: Ranking dos 10 produtos mais vendidos por volume.

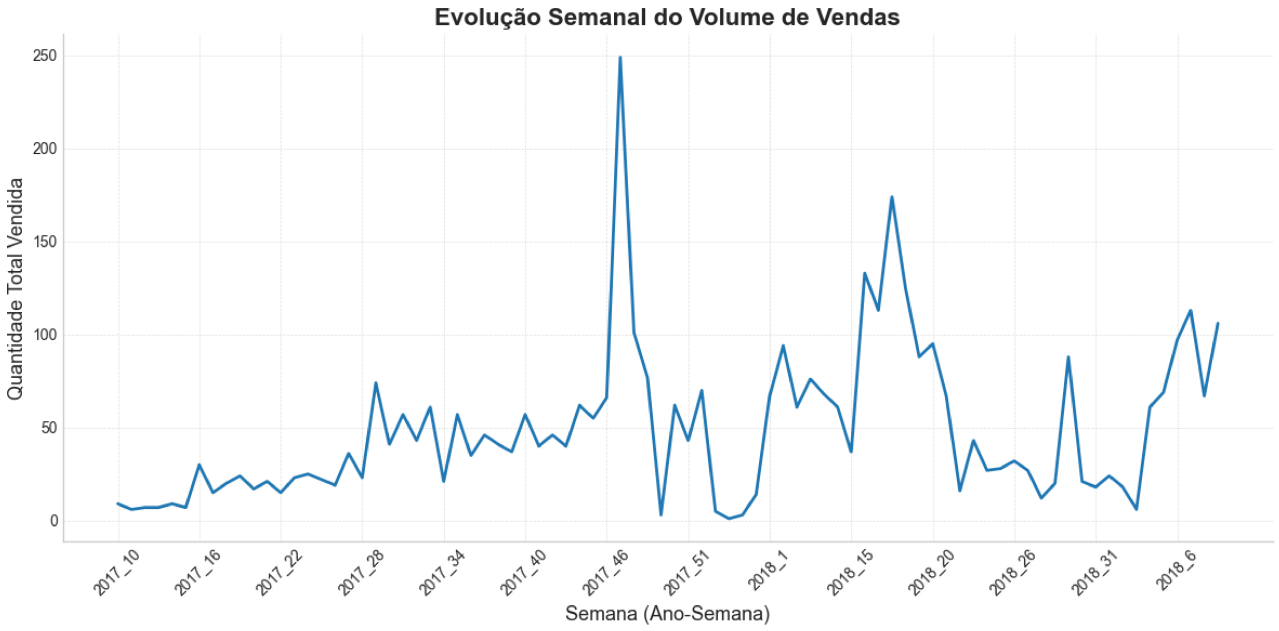

Figura 4.2: Evolução semanal do volume de vendas.

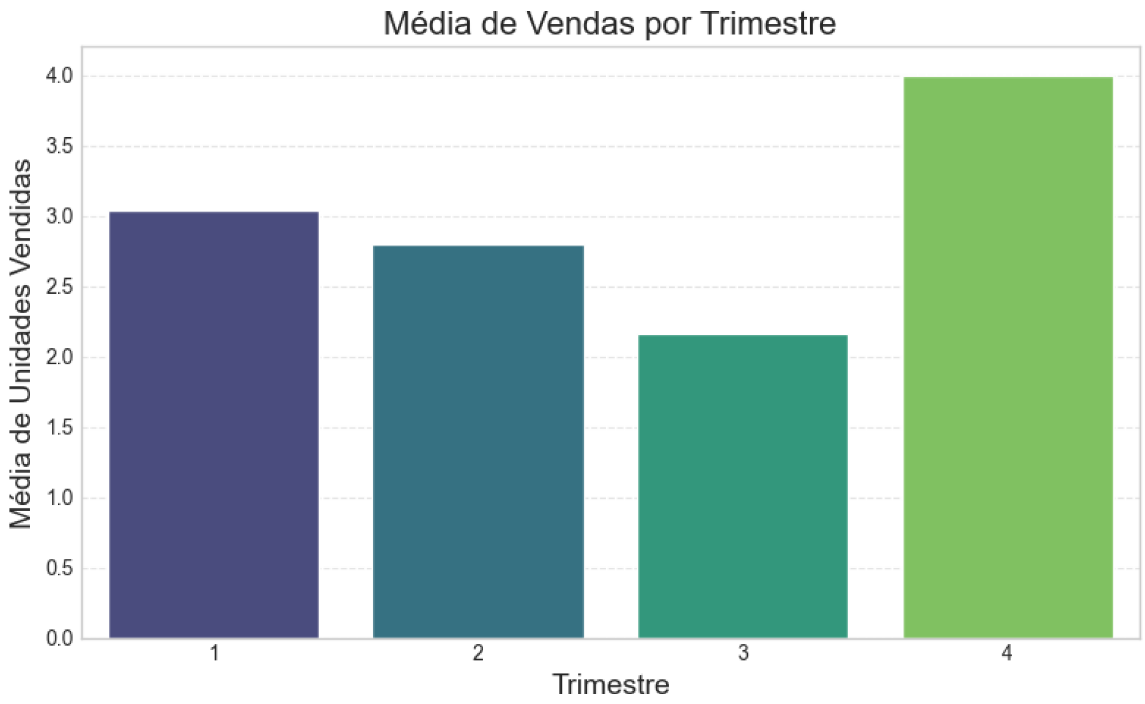

Figura 4.3: Média de vendas por trimestre.

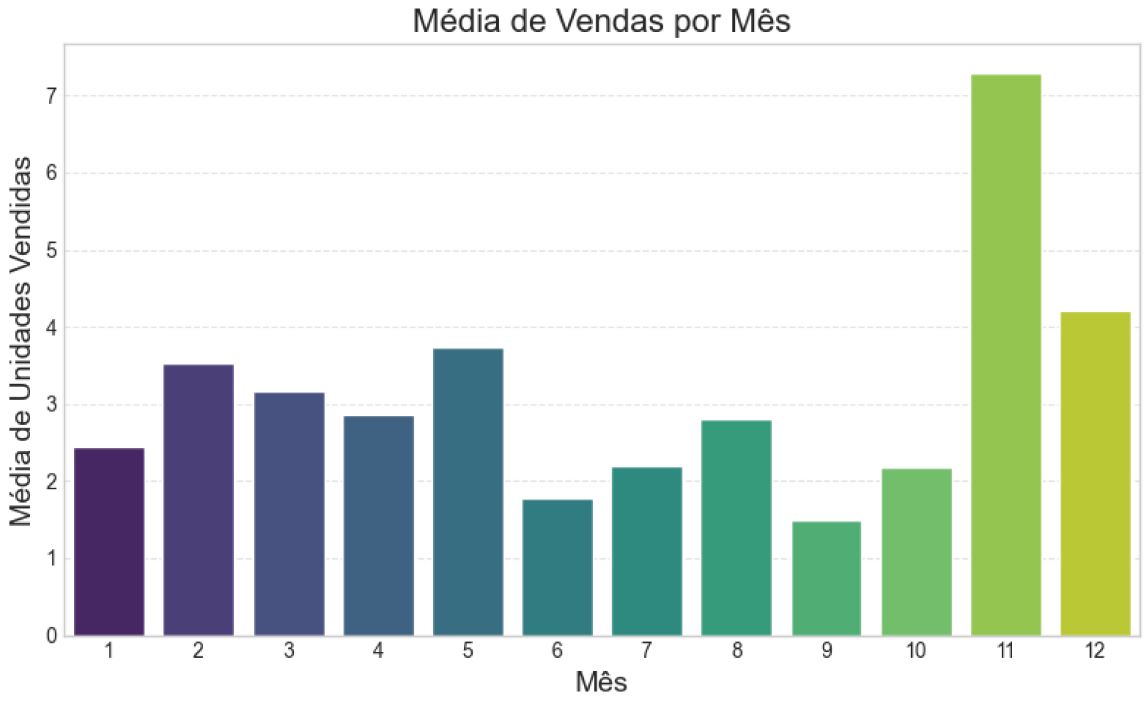

Figura 4.4: Média de vendas por mês.

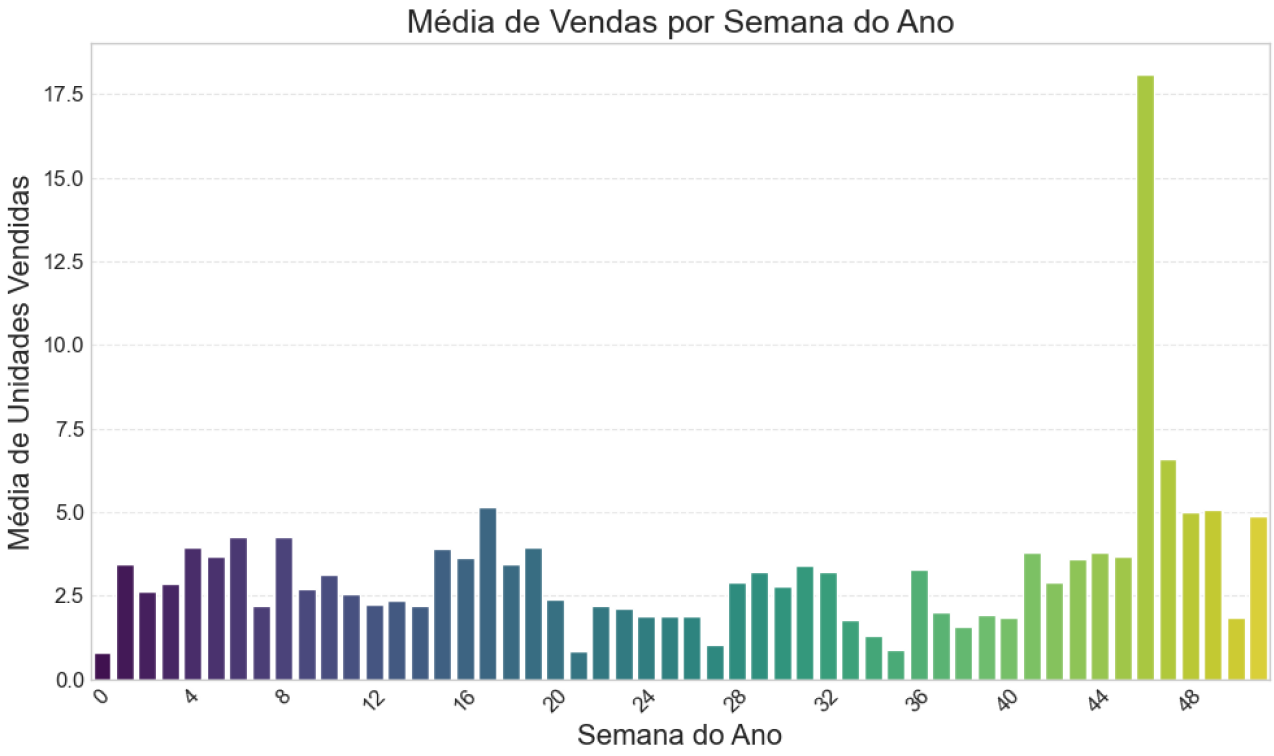

Figura 4.5: Média de vendas por semana do ano.

De forma geral, observa-se um aumento expressivo no volume de vendas ao longo do quarto trimestre, com destaque para o mês de novembro. Esse comportamento sugere um possível adiantamento das compras relacionadas às festas de fim de ano, impulsionado por eventos promocionais, como a Black Friday que ocorre no mês 11, reforçando a importância de incorporar variáveis sazonais nos modelos preditivos.

Análise das Variáveis Criadas

Nota Média de Avaliação

A nota média de avaliação dos produtos reflete a percepção dos consumidores sobre a qualidade e a satisfação com o produto. Essa variável pode ser um indicador importante da confiança e da preferência do consumidor. A Figura 4.6 apresenta a relação entre a nota média e o volume de vendas.

A Figura 4.6 sugere que a nota média por si só não é o fator determinante para o volume de vendas. Há grande variabilidade na quantidade vendida para cada faixa de nota. Embora a maioria dos produtos venda pouco, há exceções com volumes muito altos, independentemente da nota média, indicando que outros fatores também influenciam a demanda.

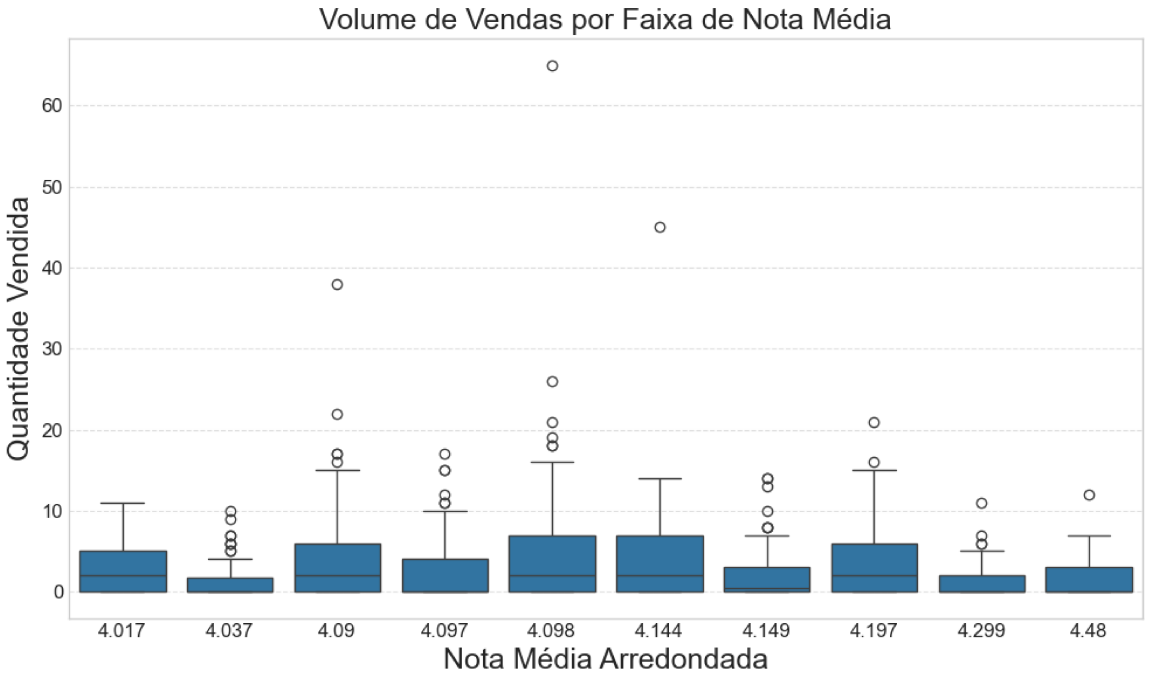

Figura 4.6: Volume de vendas por faixa de nota média.

Preço Médio do Produto

O preço médio pode influenciar diretamente a decisão de compra dos consumidores, atuando como um fator de atratividade para o volume de vendas. A Figura 4.7 apresenta essa relação.

A Figura 4.7 sugere que a relação entre preço médio e volume de vendas não é linear. Produtos em diferentes faixas de preço podem alcançar volumes variados, incluindo volumes altos, o que indica a influência de outros fatores além do preço.

Tempo Médio de Entrega

O tempo médio de entrega é um indicador da eficiência logística, que pode afetar a satisfação do cliente e a competitividade do produto. A Figura 4.8 mostra a relação entre o tempo de entrega médio e o volume de vendas.

A Figura 4.8 não mostra uma relação inversa linear clara (ou seja, de que quanto menor o tempo, maior a venda). Curiosamente, um tempo de entrega ligeiramente mais longo (como 12,19 dias) está associado a uma distribuição de vendas mais alta em comparação com prazos extremamente curtos. Esse padrão reflete características logísticas da plataforma e não necessariamente uma desvantagem comercial.

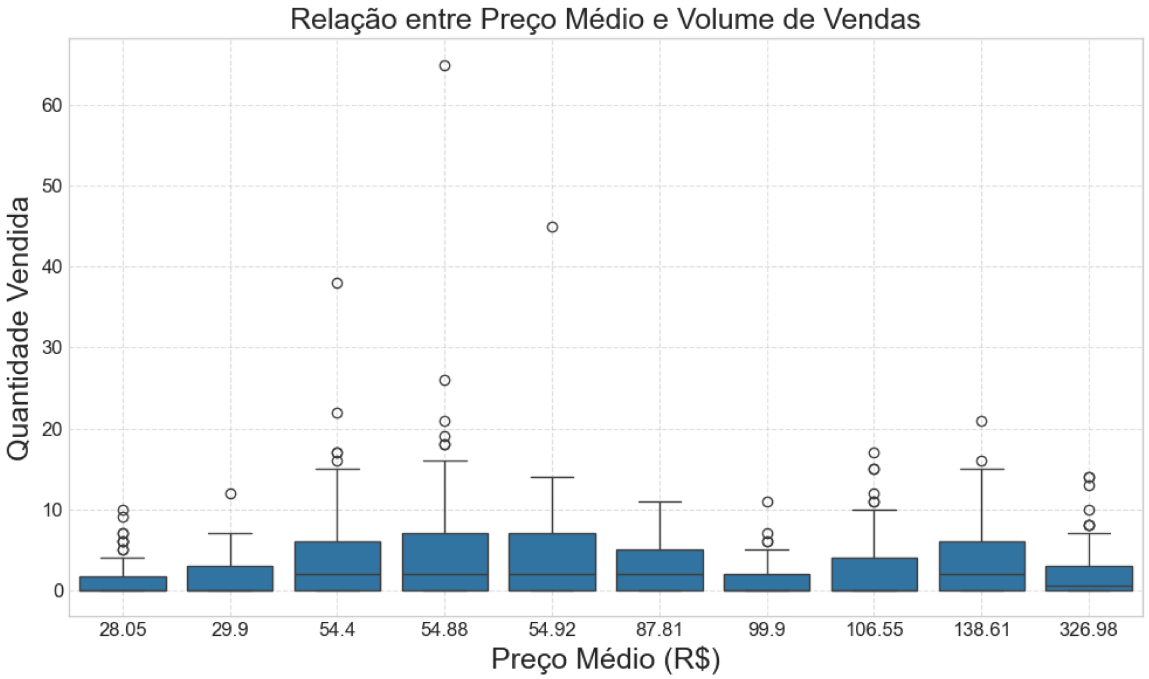

Figura 4.7: Relação entre preço médio e volume de vendas.

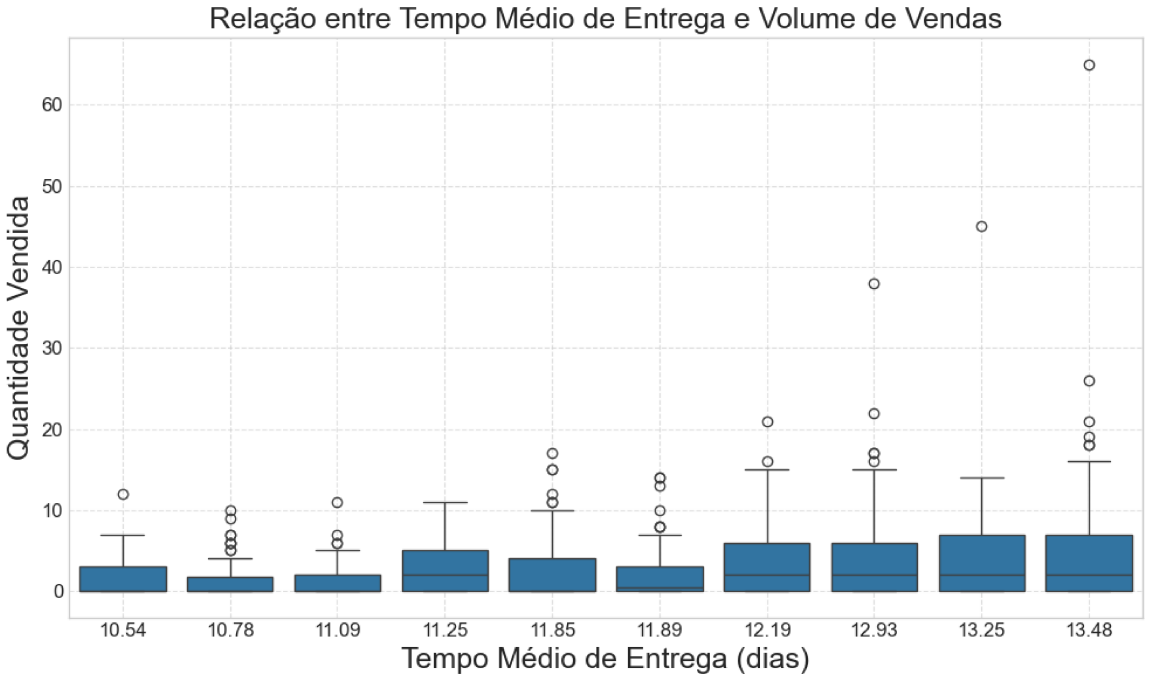

Figura 4.8: Relação entre tempo médio de entrega e volume de vendas.

4.2 RESULTADOS POR ALGORITMO

Para todos os algoritmos abordados neste trabalho, foi ajustado um modelo individual para cada produto. Essa abordagem permite capturar com maior precisão os padrões específicos de comportamento de vendas de cada item, respeitando as particularidades de suas séries temporais, além de possibilitar o uso de algoritmos diferentes para produtos distintos, conforme suas características e desempenho preditivo.

4.2.1 SARIMA

Tabela 4.1: Diagnóstico dos modelos SARIMA para os produtos.

Produto	Parâmetros	p-valores		
		LjungBox_p	Normalidade_p	ADF_p
1	(2,1,3)(0,0,0,52)	0,7330	0,0000	0,0000
2	(0,0,0)(0,1,0,52)	0,0000	0,0000	0,1715
3	(2,1,1)(1,0,0,52)	0,0000	0,0000	0,0047
4	(0,0,0)(0,1,0,52)	0,0065	0,0000	0,0005
5	(0,1,2)(0,0,0,52)	0,5442	0,0000	0,6713
6	(1,1,1)(0,0,0,52)	0,2717	0,0000	0,4119
7	(2,1,1)(0,0,0,52)	0,2001	0,0000	0,9453
8	(2,1,0)(0,0,0,52)	0,1577	0,0000	0,9739
9	(2,0,0)(0,1,0,52)	0,6693	0,0016	0,0315
10	(0,1,1)(0,0,0,52)	0,4537	0,0000	0,3611

Conforme pode ser observado na Tabela 4.1, em praticamente todas as séries temporais os resíduos apresentaram p-valores próximos de zero no teste de normalidade, indicando violação da hipótese de normalidade. Embora essa violação não comprometa necessariamente o desempenho preditivo dos modelos, ela deve ser considerada com cautela em análises inferenciais.

Outro ponto relevante é que algumas séries, como as dos produtos 2, 5, 6, 7, 8 e 10, permaneceram não estacionárias mesmo após as diferenciações impostas automaticamente pelo procedimento de auto ARIMA. [2].

Em particular, o modelo do produto 7 apresentou indícios de *subajuste*. O subajuste ocorre quando o modelo é demasiadamente simples para capturar a dinâmica dos dados, resultando em resíduos com padrões não explicados e falha em representar adequadamente a estrutura temporal da série. Esse problema foi evidenciado pela não estacionariedade persistente da série (ADF p-valor = 0,9453) e pela qualidade limitada das previsões. Apesar da possibilidade de reparametrização, optou-se por manter o modelo para garantir consistência e comparabilidade entre os produtos.

Como ilustração, a equação estimada para o produto 8, cujo modelo selecionado foi um ARIMA(2,1,0)(0,0,0), pode ser escrita como 4.1:

$$\nabla^1 \Delta^0 Z_t = 0,3842\, Z_{t-1} + 0,4809\, Z_{t-2} + \varepsilon_t \tag{4.1}$$

O código utilizado para o ajuste dos modelos SARIMA encontra-se no Apêndice A.1.

4.2.2 Regressão Ridge

A Tabela 4.2 apresenta um recorte das variáveis com maior colinearidade identificada pela métrica de Variance Inflation Factor (VIF). Nota-se que diversas variáveis dummificadas de tempo (semanas e ano) e também feriados apresentam valores de VIF infinitos, o que indica dependência linear perfeita entre elas. Esse resultado evidencia a redundância de informação no conjunto de preditores e reforça a necessidade da utilização da regressão Ridge para mitigar os efeitos da multicolinearidade.

Tabela 4.2: Exemplo de variáveis com multicolinearidade severa (VIF = ∞).

Variável	**VIF**
SEMANA_ANO_27	∞
SEMANA_ANO_39	∞
SEMANA_ANO_47	∞
SEMANA_ANO_43	∞
SEMANA_ANO_42	∞
SEMANA_ANO_41	∞
SEMANA_ANO_40	∞
ANO_2017	∞
SEMANA_ANO_5	∞
Feriado_Black Friday	∞

Neste trabalho, os valores de λ utilizados na regressão Ridge foram definidos individualmente para cada produto, com base em testes preliminares que buscaram equilibrar a redução do sobreajuste e a manutenção da capacidade preditiva. Essa abordagem permite ajustar a penalização de acordo com a variabilidade e a estrutura de cada série temporal, evitando coeficientes excessivamente amplificados ou regularização excessiva que prejudicaria o desempenho do modelo. Os valores definidos para cada modelo são apresentados na tabela 4.3

A equação 4.2 representa um exemplo de predição gerada. Esse modelo foi ajustado individualmente para o produto 8, considerando variáveis sazonais e feriados. A regressão Ridge mantém a estrutura da regressão linear, mas aplica uma penalização sobre os coeficientes para evitar sobreajuste, principalmente em situações de multicolinearidade ou conjuntos de dados pequenos.

Tabela 4.3: Valores de λ definidos para cada produto.

Produto	Valor de λ
1	95,0724
2	95,0724
3	98,6897
4	95,0724
5	37,4550
6	37,4550
7	5,8094
8	37,4550
9	37,4550
10	37,4550

$$\begin{aligned}\hat{y} = {} & 5,7511 - 0,0693\,X_0 + 0,0404\,X_5 - 0,0952\,X_6 - 0,0254\,X_7 - 0,0304\,X_8 + 0,0063\,X_9 \\ & - 0,0701\,X_{10} - 0,0012\,X_{11} - 0,0339\,X_{12} - 0,0122\,X_{13} - 0,0176\,X_{14} \\ & - 0,0785\,X_{15} \cdots - 0,0701\,X_{66} + 0,1625\,X_{67} \cdots + 0,2603\,X_{69} \\ & - 0,0888\,X_{70} \cdots - 0,0178\,X_{81} + 0,2427\,X_{82} \cdots - 0,0491\,X_{85} \end{aligned} \tag{4.2}$$

Os coeficientes completos para cada variável podem ser consultados na Tabela 3.2.

Observa-se que o número de semanas desde o início da série temporal (X_0) apresenta coeficiente negativo, sugerindo uma tendência de leve diminuição nas vendas ao longo do tempo. Entre os feriados, apenas Black Friday (X_5) apresenta coeficiente positivo, indicando um aumento esperado de vendas nessa semana específica, enquanto a maioria dos outros feriados apresenta coeficientes negativos, sugerindo que essas datas não tiveram efeito de incremento nas vendas para este produto.

O código utilizado para o ajuste dos modelos de Regressão Ridge encontra-se no Apêndice B.1.

4.2.3 Modelo LightGBM(LGBM)

A Tabela 4.4 apresenta os valores dos hiperparâmetros utilizados nos modelos LGBM ajustados individualmente para cada produto, juntamente com as principais métricas de desempenho obtidas. Estes hiperparâmetros foram ajustados conforme metodologia apresentada na Subseção 3.4.3. Observa-se que não houve uma configuração única capaz de capturar as características de todas as séries temporais, sendo necessária a adaptação individual.

Como pode ser observado, houve variação relevante na configuração dos modelos, com alguns produtos ajustados por meio de taxas de aprendizado mais elevadas e profundidades reduzidas (como os produtos 1, 3, 5, 6 e 10), enquanto outros exigiram árvores mais profundas ou menor taxa de aprendizado (como os produtos 2, 7, 8 e 9). Essa diversidade reforça a heterogeneidade do comportamento das séries temporais analisadas e a necessidade de ajustes específicos para

Tabela 4.4: Hiperparâmetros e métricas dos modelos LGBM ajustados individualmente por produto.

Produto	N° de folhas	Profundidade	Taxa de apredizado	N° de estimadores	Amostras mínimas por folha
1	24	3	0,2525	179	25
2	25	6	0,0269	179	10
3	24	3	0,2525	179	25
4	24	3	0,2525	179	25
5	24	3	0,2525	179	25
6	24	3	0,2525	179	25
7	19	6	0,0823	57	11
8	14	8	0,2366	103	10
9	6	7	0,1803	93	22
10	24	3	0,2525	179	25

cada caso.

O código utilizado para o ajuste dos modelos de LightGBM(LGBM) encontra-se no Apêndice C.1.

4.2.4 Rede Neural Recorrente (LSTM)

A Tabela 4.5 apresenta os valores dos hiperparâmetros utilizados nos modelos LSTM ajustados individualmente para cada produto, evidenciando a diversidade estrutural adotada para melhor capturar as características específicas de cada série temporal.

Tabela 4.5: Hiperparâmetros dos modelos LSTM ajustados individualmente por produto.

Produto	N° de unidades ocultas	N° camadas	Taxa de abandono	Taxa de aprendizado
1	64	1	0,5	0,0005
2	128	3	0,0	0,001
3	16	3	0,2	0,005
4	128	3	0,3	0,005
5	32	1	0,0	0,001
6	128	3	0,3	0,005
7	64	2	0,2	0,001
8	64	2	0,0	0,0005
9	128	2	0,2	0,0001
10	32	2	0,5	0,005

Como pode ser observado, a arquitetura dos modelos varia significativamente entre os produtos. Produtos como o 2, 4 e 6 exigiram mais camadas e maior número de unidades ocultas, refletindo a complexidade ou maior variabilidade das séries temporais correspondentes e a necessidade de uma adaptação individual às particularidades de cada série temporal, como sazonalidade, variância e quantidade de dados disponíveis.

O código utilizado para o ajuste dos modelos de rede neural recorrente (LSTM) encontra-se no Apêndice D.1.

4.3 Resultados Gerais

Nesta seção, são apresentados os resultados obtidos com os modelos ajustados para previsão do volume de vendas semanais por produto. Avaliamos quatro algoritmos distintos: SARIMA, Regressão Ridge, LightGBM e LSTM, considerando como métricas de desempenho o Erro Ab-

soluto Médio (MAE), a Raiz do Erro Quadrático Médio (RMSE) e o Erro Percentual Absoluto Simétrico (sMAPE).

Para o Produto 1 (Figura 4.9), o modelo com melhor desempenho foi a LSTM, apresentando os menores valores de MAE (1,64) e RMSE (1,95), indicando boa capacidade de ajuste às variações temporais da série.

No Produto 2 (Figura 4.10), a LSTM novamente apresentou o menor erro (MAE = 1,09; RMSE = 1,48), superando os demais modelos, especialmente a Regressão Linear, que obteve erros significativamente mais altos.

Para o Produto 3 (Figura 4.11), o melhor desempenho foi obtido pelo LightGBM, com MAE = 2,55 e RMSE = 3,38, demonstrando robustez na previsão em relação aos demais modelos, que apresentaram maior variabilidade nos erros.

No Produto 4 (Figura 4.12), a LSTM apresentou novamente o melhor resultado, com MAE = 1,64 e RMSE = 2,63, destacando-se frente aos demais modelos, cujos erros foram consideravelmente maiores.

O Produto 5 (Figura 4.13) apresentou um resultado equilibrado entre os modelos LightGBM, Regressão Ridge e SARIMA, todos com erros semelhantes (MAE $\approx$ 1,9–2,2). Nesse caso, não houve um modelo claramente superior, sugerindo que a série apresenta padrão simples e previsível.

Para o Produto 6 (Figura 4.14), o SARIMA obteve o melhor desempenho (MAE = 2,55; RMSE = 3,28), superando os modelos de aprendizado de máquina, o que indica que a série pode seguir um comportamento mais estacionário e linear.

O Produto 7 (Figura 4.15) teve como destaque a LSTM, com MAE = 1,64 e RMSE = 2,00, demonstrando boa capacidade de capturar padrões temporais, enquanto os modelos tradicionais apresentaram erros superiores.

No Produto 8 (Figura 4.16), o LightGBM apresentou o menor erro quadrático médio (RMSE = 5,04) e desempenho competitivo em relação às demais métricas, sendo o modelo mais adequado para essa série.

Para o Produto 9 (Figura 4.17), o SARIMA obteve os menores valores de MAE (3,18) e RMSE (4,03), superando os modelos de aprendizado de máquina, o que reforça o bom desempenho de abordagens tradicionais em certas séries.

Por fim, no Produto 10 (Figura 4.18), o LightGBM apresentou o melhor resultado (MAE = 1,73; RMSE = 2,15), seguido de perto pela Regressão Linear e pelo SARIMA, que obtiveram valores idênticos de erro.

De modo geral, embora os modelos de aprendizado de máquina tenham apresentado bom desempenho em várias séries, os modelos tradicionais como o SARIMA também se mostraram competitivos em alguns casos. Isso evidencia que, apesar do crescente uso de técnicas de machine learning, modelos clássicos ainda podem oferecer resultados mais consistentes para séries temporais com comportamento regular ou sazonal bem definido.

A Tabela 4.6 resume o desempenho de cada modelo para os 10 produtos analisados. A seguir, apresentamos os gráficos comparando os valores reais com as previsões feitas por cada

modelo ao longo do tempo. O cálculo das métricas e a criação dos gráficos de previsões foram obtidos com o código apresentado no Apêndice E.1.

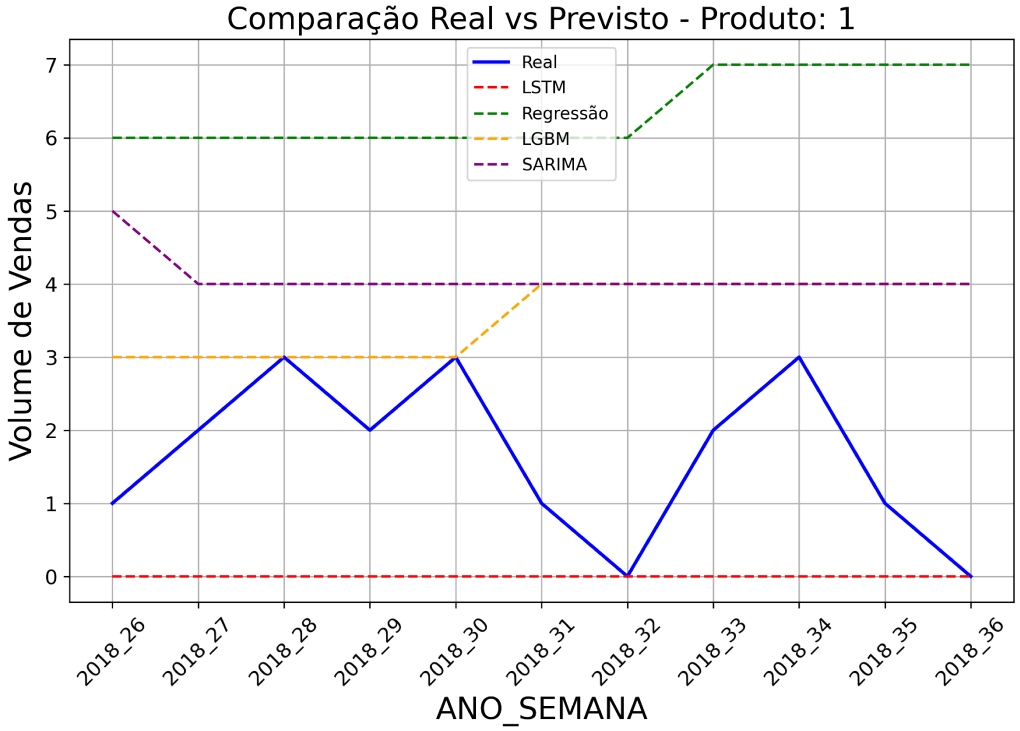

Figura 4.9: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 1.

Tabela 4.6: Desempenho dos modelos.

Produto	Modelo	MAE	RMSE	sMAPE
1	LightGBM	1,91	2,35	83,20
	LSTM	1,64	1,95	163,64
	Ridge	4,73	4,88	123,65
	SARIMA	2,45	2,71	96,28
2	LightGBM	3,91	4,03	136,36
	LSTM	1,09	1,48	145,45
	Ridge	9,64	9,70	165,36
	SARIMA	3,55	4,19	135,83
3	LightGBM	2,55	3,38	81,83
	LSTM	2,64	3,21	145,97
	Ridge	6,82	7,22	118,44
	SARIMA	4,27	4,81	102,36
4	LightGBM	8,18	8,70	145,98
	LSTM	1,64	2,63	124,24
	Ridge	11,55	11,86	157,25
	SARIMA	7,18	9,48	126,96
5	LightGBM	1,91	2,15	92,16
	LSTM	2,36	3,13	145,45
	Ridge	1,91	2,15	92,16
	SARIMA	2,18	2,63	90,68
6	LightGBM	2,45	2,81	96,71
	LSTM	3,36	4,32	163,64
	Ridge	2,45	2,81	96,71
	SARIMA	2,55	3,28	83,33
7	LightGBM	5,36	5,55	132,85
	LSTM	1,64	2,00	162,42
	Ridge	5,55	5,79	132,85
	SARIMA	6,36	6,52	139,17
8	LightGBM	4,09	5,04	108,97
	LSTM	4,18	6,15	142,99
	Ridge	4,36	5,13	104,95
	SARIMA	5,45	6,05	110,80
9	LightGBM	5,55	5,81	121,40
	LSTM	2,18	2,45	200,00
	Ridge	8,27	8,44	138,11
	SARIMA	3,18	4,03	91,66
10	LightGBM	1,73	2,15	71,26
	LSTM	2,55	3,41	121,21
	Ridge	2,00	2,41	73,18
	SARIMA	2,00	2,41	73,18

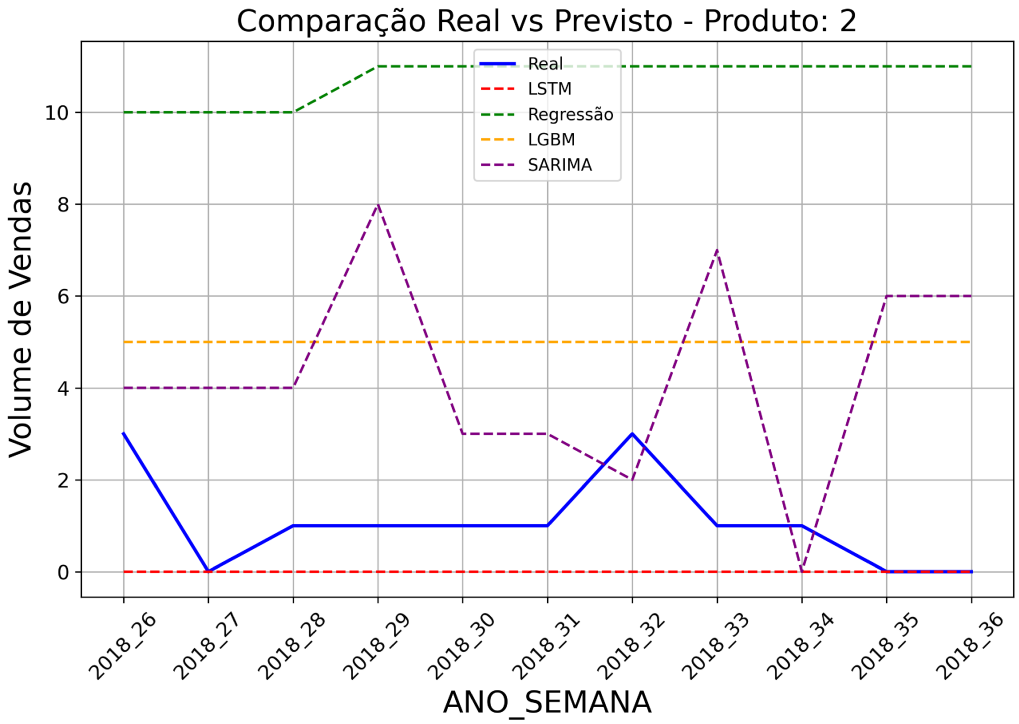

Figura 4.10: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 2.

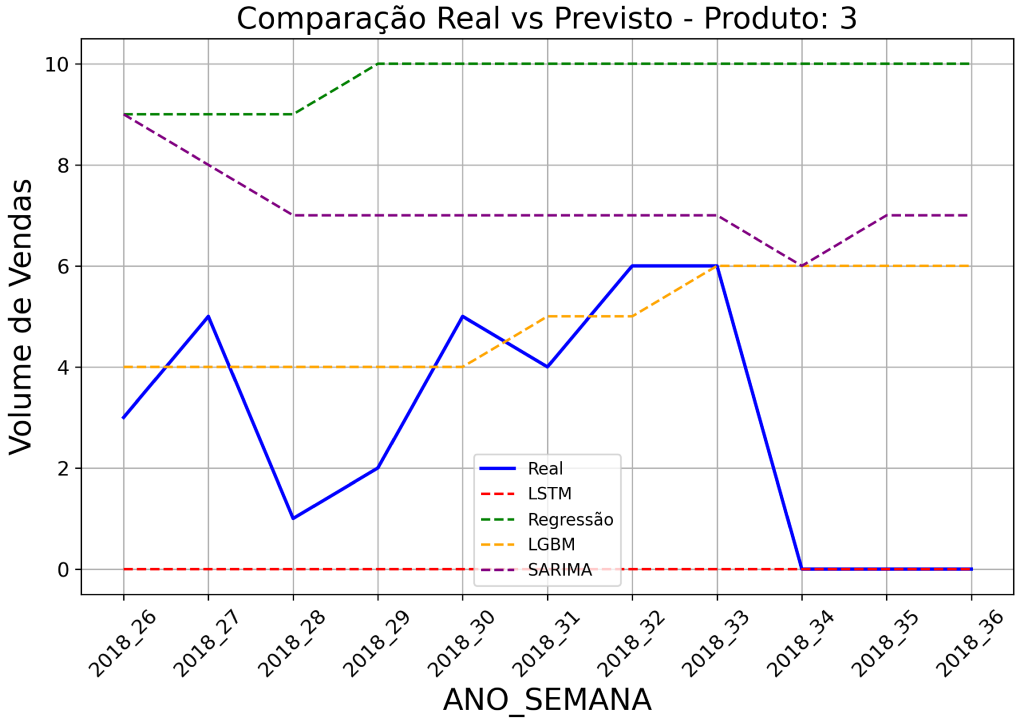

Figura 4.11: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 3.

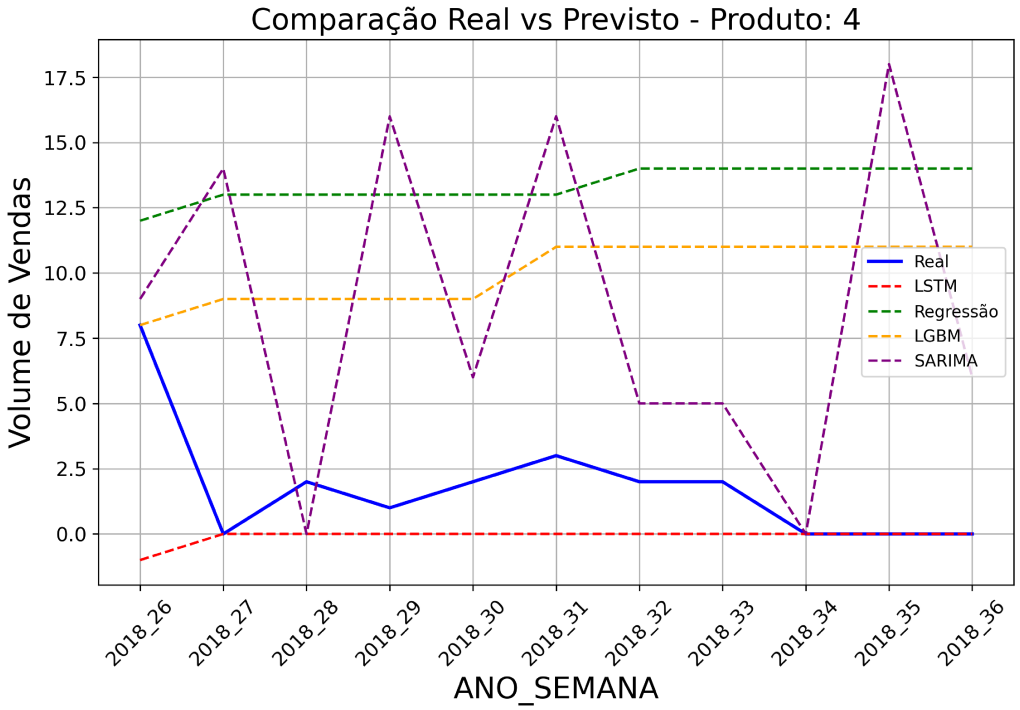

Figura 4.12: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 4.

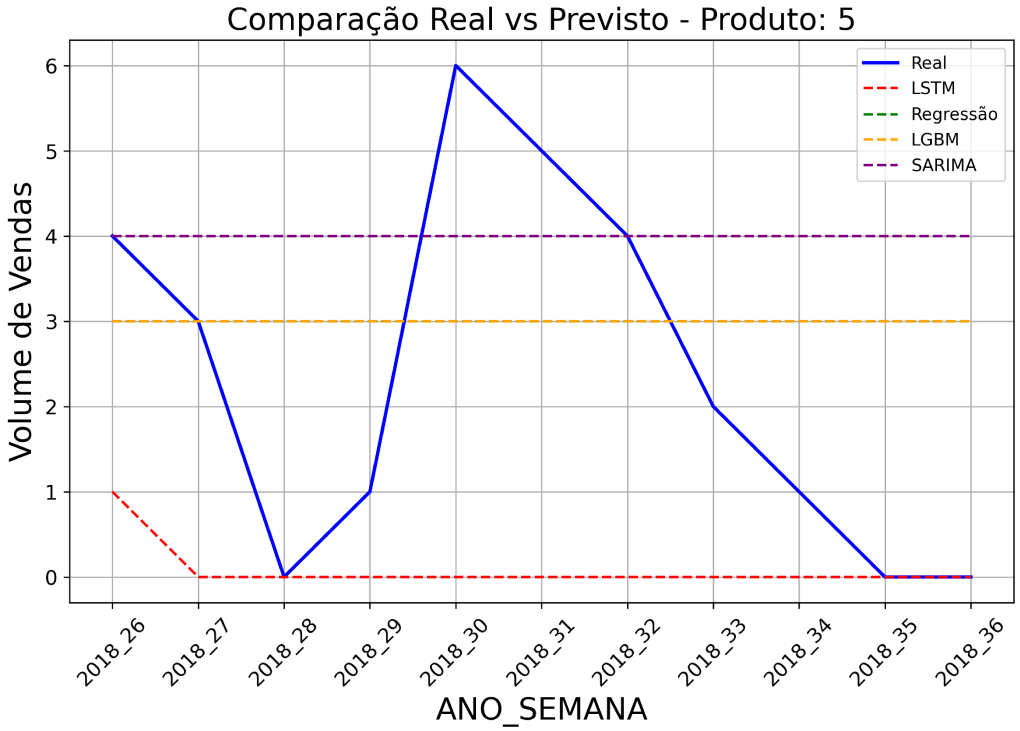

Figura 4.13: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 5.

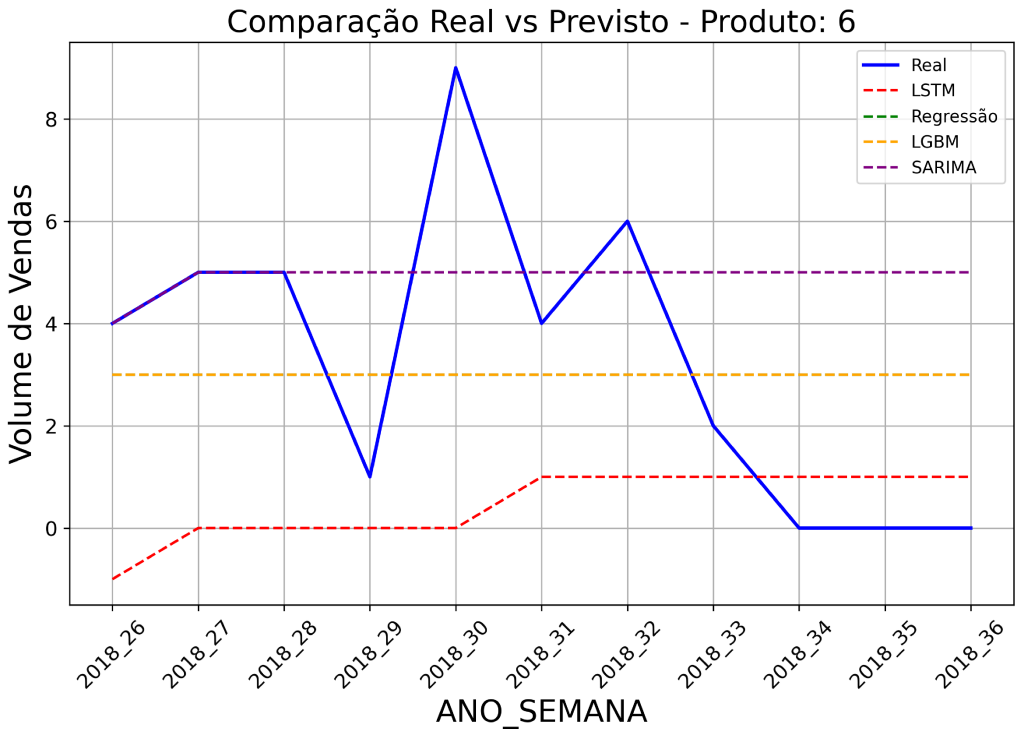

Figura 4.14: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 6.

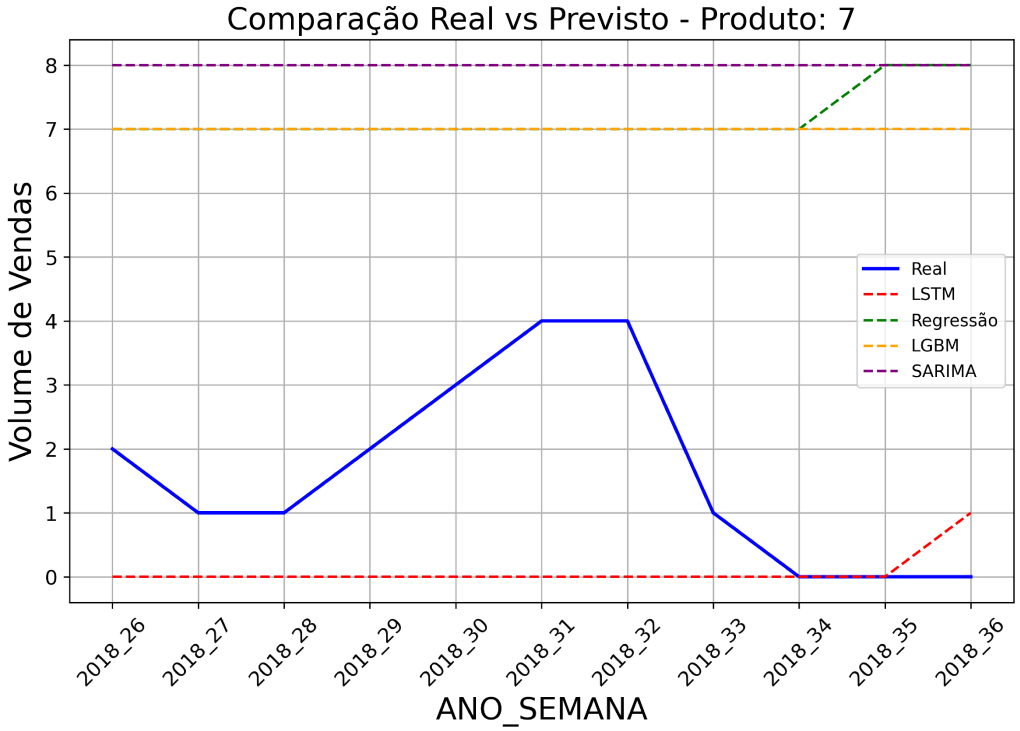

Figura 4.15: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 7.

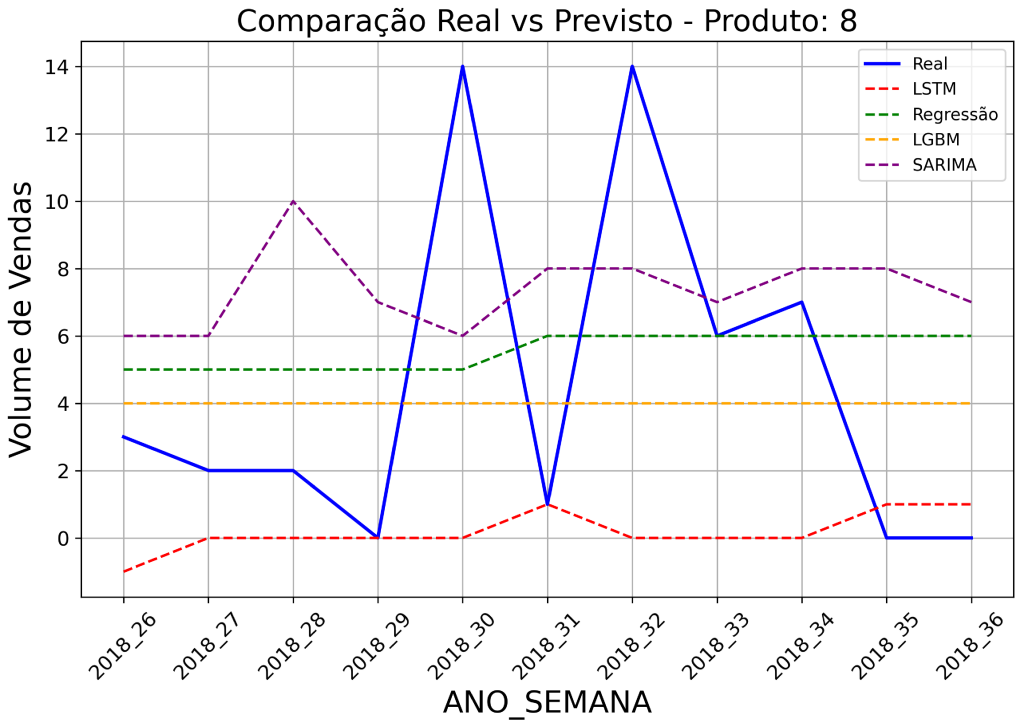

Figura 4.16: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 8.

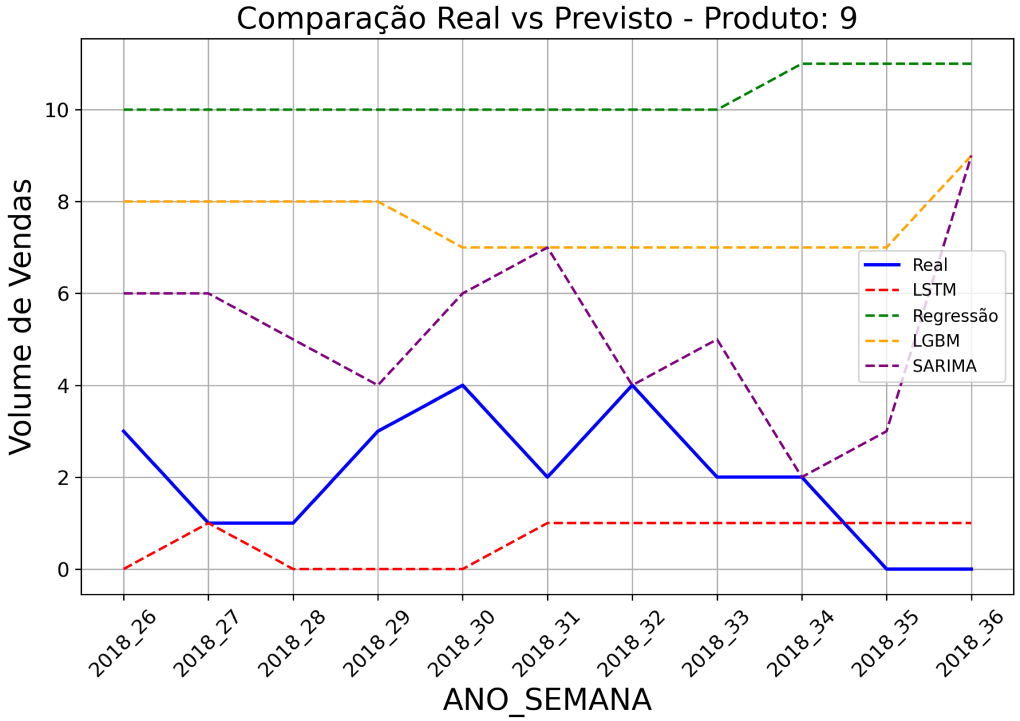

Figura 4.17: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 9.

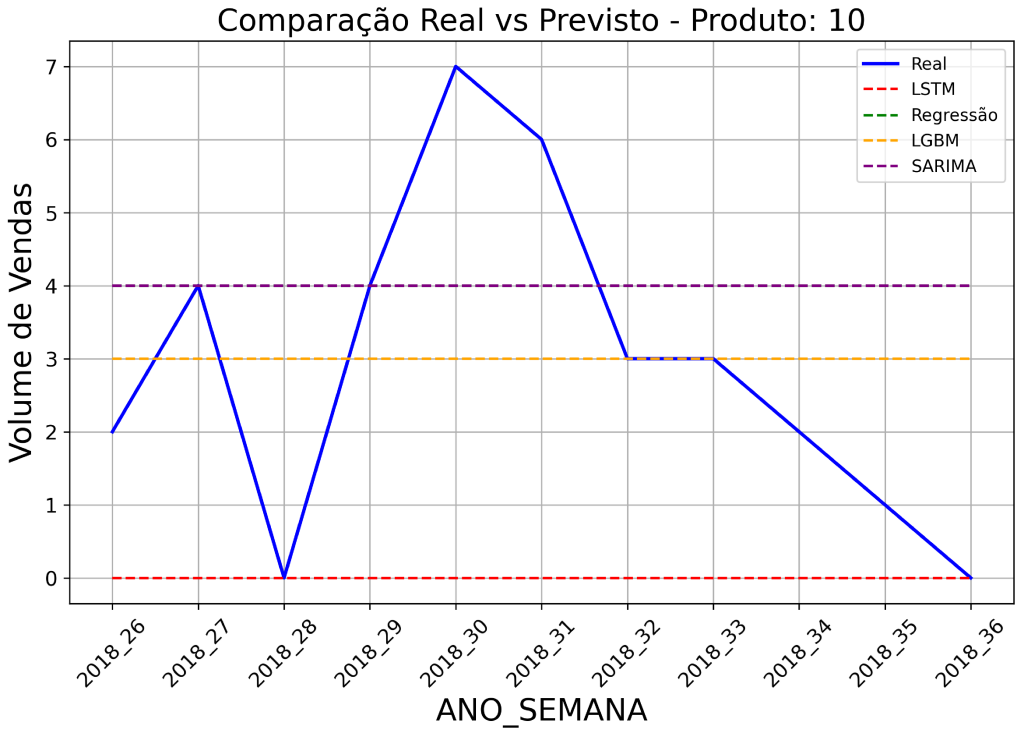

Figura 4.18: Previsão do volume de vendas, para o período compreendido da 26ª semana à 36ª de 2018, do produto 10.

5. Conclusões

A pesquisa permitiu verificar que o treinamento dos modelos SARIMA, Regressão ridge, LightGBM e LSTM, na linguagem python, é de implementação relativamente fácil e apresenta pouca demanda computacional atualmente. A partir da comparação dos modelos observou-se que a rede neural LSTM foi mais eficiente para a maioria dos produtos e em segundo lugar, o LGBM. No entanto, o modelo SARIMA também se sobressaiu em alguns casos, dependendo da métrica utilizada na comparação, demonstrando que a modelagem tradicional não pode ser descartada.

De maneira geral, os resultados obtidos indicaram que a tarefa de previsão de vendas em marketplaces é desafiadora, principalmente devido à alta variabilidade e ao comportamento irregular das séries temporais. Nenhum dos modelos analisados apresentou desempenho consistentemente satisfatório em todos os produtos. A maior parte dos modelos, mostrou-se limitado, pois não conseguiu capturar a variação temporal das séries, produzindo previsões praticamente estáticas.

Um fator crítico identificado é a demanda historicamente esparsa de muitos produtos, com grande número de semanas sem vendas ou com volumes muito baixos. Essa característica torna o problema ainda mais desafiador, pois reduz a capacidade dos modelos de identificar padrões confiáveis e aumenta a variabilidade das métricas de erro. Além disso, a alta heterogeneidade entre produtos, sazonalidade irregular e presença de eventos promocionais dificultam a generalização de qualquer modelo único para todo o conjunto de produtos.

Outro ponto importante foi a análise exploratória das variáveis criadas, que mostrou relações parciais entre características como sazonalidade, quantidade de vendedores tempo de entrega dos produtos e períodos promocionais (exemplo da Black Friday) com o volume de vendas. Esses resultados reforçam a complexidade do problema e a necessidade de metodologias mais avançadas para captar adequadamente a dinâmica da demanda.

Para trabalhos futuros, recomenda-se ampliar a base de dados para períodos mais recentes, explorar variáveis externas (como indicadores econômicos ou estratégias de marketing) e a avaliação de arquiteturas mais avançadas de séries temporais, como modelos baseados em Transformers aplicados a séries temporais, que possuem potencial para capturar dependências de longo prazo, mas demandam maior volume de dados e a previsão hierárquica, que explora diferentes níveis de agregação (produto, categoria, total) além de explorar modelos e metodologias especificamente desenvolvidos para lidar com demanda intermitente, aumentando a consistência das previsões. Essas alternativas, embora promissoras, não foram exploradas neste

estudo por ampliarem significativamente a complexidade metodológica e computacional, mas representam caminhos relevantes para pesquisas futuras.

Em síntese, este estudo reforça que a previsão de vendas em marketplaces é um problema de elevada complexidade. E que, embora os modelos analisados não tenham alcançado previsões robustas de forma geral, eles fornecem uma base inicial para investigações futuras e destacam o potencial do uso de aprendizado de máquina em aplicações comerciais. Assim, o trabalho contribui tanto para a compreensão dos desafios inerentes à previsão de vendas quanto para o direcionamento de pesquisas futuras na área.

Referências Bibliográficas

[1] *Brazilian E-Commerce Public Dataset by Olist*, 2021. https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce.

[2] Taylor G. Smith et al.: *pmdarima.arima.auto_arima — automatic order selection for ARIMA models*. https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html, 2023. Acesso em: 19 set. 2025.

[3] Andrea: *Árvores de Regressão*, 2022. https://medium.com/@andreaselias/%C3%A1rvores-de-regress%C3%A3o-42f703026d2c, acessado em 02/11/2024.

[4] Awari: *O Guia Definitivo para Feature Engineering*, 2023. https://awari.com.br/o-guia-definitivo-para-feature-engineering/.

[5] Guilherme Barbosa, Govinda Bezerra, Dianne Medeiros, Martin Andreoni e Diogo Menezes: *Segurança em Redes 5G: Oportunidades e Desafios em Detecção de Anomalias e Predição de Tráfego Baseadas em Aprendizado de Máquina*, páginas 145–189. ResearchGate, outubro 2021, ISBN 9786587003658.

[6] Anna C. Barros, Daiane Marcolino de Mattos, Ingrid Christyne Luquett de Oliveira e outros: *Análise de Séries Temporais em R: Curso Introdutório*. GEN Atlas, Rio de Janeiro, 2017, ISBN 9788595154902. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/reader/books/9788595154902/. Acesso em: 13 jul. 2025.

[7] R. Charnet, C. d. L. Freire, E. M. R. Charnet e H. Bonvino: *Análise de modelos de regressão linear com aplicações*. Unicamp, Campinas, São Paulo, 1999.

[8] Google for Developers: *Gradient Boosted Decision Trees*, 2024. https://developers.google.com/machine-learning/decision-forests/intro-to-gbdt, acessado em 02/11/2024.

[9] scikit-learn developers: *MinMaxScaler*. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html, 2025. Acesso em: 19 set. 2025.

[10] Pakorn Ditthakit, Sirimon Pinthong, Nureehan Salaeh, Jakkarin Weekaew e Quoc Pham: *Comparative study of machine learning methods and GR2M model for monthly runoff prediction*. Ain Shams Engineering Journal, 14, setembro 2022.

[11] Junliang Fan, Xin Ma, Lifeng Wu, Fucang Zhang, Xiang Yu e Wenzhi Zeng: *Light Gradient Boosting Machine: An efficient soft computing model for estimating daily reference evapotranspiration with local and external meteorological data*. Agricultural Water Management, 225, 2019, ISSN 0378-3774. https://www.sciencedirect.com/science/article/pii/S0378377419302768.

[12] Forbes: *Predicting The Future Of Demand: How Amazon Is Reinventing Forecasting With Machine Learning*, 2021. https://www.forbes.com/sites/amazonwebservices/2021/12/03/predicting-the-future-of-demand-how-amazon-is-reinventing-forecasting-with-machine-learning/, acessado em 12/10/2024.

[13] Wayne A. Fuller: *Introduction to Statistical Time Series*. Wiley, New York, 2nd ed. edição, 1996, ISBN 9780471552390.

[14] A. Géron: *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn, Keras & TensorFlow*. Alta Books, 2021, ISBN 9786555208146. https://books.google.com.br/books?id=JUBIEAAAQBAJ.

[15] Gil Giardelli: *O impulso do e-commerce no Brasil: perspectivas até 2027.*, 2024. https://www.ecommercebrasil.com.br/artigos/o-impulso-do-e-commerce-no-brasil-perspectivas-ate-2027, acessado em 12/10/2024.

[16] Instituto Federal de Goiás Câmpus Jataí: *O desafio da Rede Neural Artificial (RNA)*, 2025. https://ifgjatai.webcindario.com/rna.html, acessado em 2025-09-16.

[17] J. F. Hair, W. C. Black, B. J. Babin, R. E. Anderson e R. L. Tatham: *Análise multivariada de dados*. Bookman Editora, 2009.

[18] S. Haykin: *Redes Neurais: Princípios e Prática*. Bookman Editora, 2001, ISBN 9788577800865. https://books.google.com.br/books?id=bhMwDwAAQBAJ.

[19] Sepp Hochreiter e Jürgen Schmidhuber: *Long Short-term Memory*. Neural computation, 9:1735–80, dezembro 1997.

[20] R. Hoffmann: *Análise de regressão: uma introdução à econometria*. Portal de Livros Abertos da USP, 2016.

[21] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye e Tie Yan Liu: *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. Em I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan e R. Garnett (editores): *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[22] Sungil Kim e Heeyoung Kim: *A new metric of absolute percentage error for intermittent demand forecasts*. International Journal of Forecasting, 32(3):669–679, 2016, ISSN 0169-2070. https://www.sciencedirect.com/science/article/pii/S0169207016000121.

[23] Aryel Soares Loureiro: *Previsão de séries temporais utilizando modelos ARIMA e redes neurais artificiais*. https://imef.furg.br/images/documentos/matematica-aplicada/monografias/2023-Aryel_Soares_Loureiro.pdf, 2023. Trabalho de Conclusão de Curso (Bacharelado em Matemática Aplicada), Universidade Federal do Rio Grande.

[24] D. M. Nelson: *Uso de Redes Neurais Recorrentes para Previsão de Séries Temporais Financeiras*. Tese de Mestrado, Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Belo Horizonte, 2017.

[25] redação olist: *O que é Olist? Conheça as soluções e aumente as vendas do seu negócio*, 2023. https://olist.com/blog/pt/olist/casos-de-uso/o-que-e-olist/, acessado em 12/10/2024.

[26] redação olist: *Previsão de demanda: por que é essencial nas compras?*, 2023. https://olist.com/blog/como-empreender/planejamento-estrategico/previsao-de-demanda-no-e-commerce/, acessado em 12/10/2024.

[27] Vitória Padilha: *Como utilizar IA e ML na previsão de demandas logísticas*, 2024. https://uello.com.br/blog/como-utilizar-ia-e-ml-na-previsao-de-demandas/#:~:text=Ao%20contr%C3%A1rio%20dos%20m%C3%A9todos%20tradicionais,para%20prever%20com%20alta%20precis%C3%A3o., acessado em 12/10/2024.

[28] Niklas Christoffer Petersen, Filipe Rodrigues e Francisco Camara Pereira: *Multi-output bus travel time prediction with convolutional LSTM neural network*. Expert Systems with Applications, 120:426–435, abril 2019, ISSN 0957-4174. http://dx.doi.org/10.1016/j.eswa.2018.11.028.

[29] dr prodigy e contributors: *python-holidays: Generate and work with holidays in Python*. https://github.com/dr-prodigy/python-holidays, 2025. Acesso em: 19 set. 2025.

[30] J. N Rodrigues: *Inflação no Brasil: uma aplicação de Séries Temporais e Redes Neurais Recorrentes*. Universidade Federal de Uberlândia, 2021.

[31] Suhartono Suhartono: *Time Series Forecasting by using Seasonal Autoregressive Integrated Moving Average: Subset, Multiplicative or Additive Model*. Journal of Mathematics and Statistics, 7:20–27, janeiro 2011.

[32] W. Y. Szeto, Bidisha Ghosh, Biswajit Basu e Margaret O'Mahony: *Multivariate Traffic Forecasting Technique Using Cell Transmission Model and SARIMA Model*. Journal of Transportation Engineering, 135(9):658–667, 2009. https://ascelibrary.org/doi/abs/10.1061/%28ASCE%290733-947X%282009%29135%3A9%28658%29.

[33] Roberto Torres: *How Walmart enhances its inventory, supply chain through AI*, 2022. https://www.ciodive.com/news/walmart-AI-ML-retail/638582/, acessado em 12/10/2024.

[34] Viswa: *Unveiling Decision Tree Regression: Exploring its Principles, Implementation*, 2023. https://medium.com/@vk.viswa/unveiling-decision-tree-regression-exploring-its-principles-implementation-beb882d756c6, acessado em 02/11/2024.

A. Codigo SARIMA

Apêndice A.1: Codigo para implementacao do modelo SARIMA.

```
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pmdarima import auto_arima
from statsmodels.tsa.statespace.sarimax import SARIMAX
from tqdm import tqdm
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.stats.diagnostic import acorr_ljungbox
from scipy.stats import normaltest
from statsmodels.tsa.stattools import adfuller
import seaborn as sns
import os
from scipy.stats import jarque_bera

from sklearn.metrics import mean_absolute_error, mean_squared_error,
    mean_absolute_percentage_error

# Ocultar avisos
warnings.filterwarnings("ignore")


train = pd.read_csv('dados/train_filtered.csv')
test = pd.read_csv('dados/test_filtered.csv')

train["ANO_SEMANA"] = train['ANO'].astype(str) + '_' + train['SEMANA_ANO'].
    astype(str)
test["ANO_SEMANA"] = test['ANO'].astype(str) + '_' + test['SEMANA_ANO'].
    astype(str)


df_train = train.sort_values(by=["product_id", "ANO_SEMANA"])[["ANO_SEMANA",
     "product_id", "QTD_PRODUTOS"]]
df_test = test.sort_values(by=["product_id", "ANO_SEMANA"])[["ANO_SEMANA", "
    product_id", "QTD_PRODUTOS"]]

```

```
33
34 # Criar um dicionario para armazenar os modelos e previsoes
35 model_results = {}
36
37 # Separar as series por categoria
38 produtos = df_train["product_id"].unique()
39
40 import re
41 import numpy as np
42
43 def format_coef(c):
44     return f"{c:.4f}"
45
46 def sign_str(c):
47     return "-" if c >= 0 else "+"  # note: we'll put sign explicitly before
    coef in term building
48
49 def build_sarima_latex(model_fit, order, seasonal_order, var="Z", time_idx="
    t"):
50     """
51     Retorna uma string LaTeX com a equacao SARIMA montada a partir de
    model_fit, order e seasonal_order.
52     - order: tuple (p,d,q)
53     - seasonal_order: tuple (P,D,Q,m)
54     - var: name of variable (default "Z")
55     - time_idx: time index symbol (default "t")
56     """
57     p, d, q = order
58     P, D, Q, m = seasonal_order
59
60     # try to extract params from standard attributes
61     ar_coefs = None
62     ma_coefs = None
63     sar_coefs = None
64     sma_coefs = None
65
66     if hasattr(model_fit, "arparams"):
67         ar_coefs = np.atleast_1d(model_fit.arparams) if model_fit.arparams.
    size else np.array([])
68     if hasattr(model_fit, "maparams"):
69         ma_coefs = np.atleast_1d(model_fit.maparams) if model_fit.maparams.
    size else np.array([])
70     if hasattr(model_fit, "seasonal_arparams"):
71         sar_coefs = np.atleast_1d(model_fit.seasonal_arparams) if model_fit.
    seasonal_arparams.size else np.array([])
72     if hasattr(model_fit, "seasonal_maparams"):
73         sma_coefs = np.atleast_1d(model_fit.seasonal_maparams) if model_fit.
    seasonal_maparams.size else np.array([])
74
```

```
    # fallback: parse from params names (robust to naming differences)
    if ar_coefs is None or len(ar_coefs) == 0:
        ar_list = []
        for name, val in model_fit.params.items():
            # common non-seasonal AR name patterns: 'ar.L1', 'ar.L2' or 'ar.L1'
            if re.search(r'(^|[^a-zA-Z])ar(\.|_)?L\d+$', name, re.I) and not re.search(r'seasonal', name, re.I):
                ar_list.append((int(re.findall(r'L(\d+)$', name)[0]), val))
        ar_list.sort(key=lambda x: x[0])
        ar_coefs = np.array([v for (_, v) in ar_list]) if ar_list else np.array([])

    if ma_coefs is None or len(ma_coefs) == 0:
        ma_list = []
        for name, val in model_fit.params.items():
            if re.search(r'(^|[^a-zA-Z])ma(\.|_)?L\d+$', name, re.I) and not re.search(r'seasonal', name, re.I):
                ma_list.append((int(re.findall(r'L(\d+)$', name)[0]), val))
        ma_list.sort(key=lambda x: x[0])
        ma_coefs = np.array([v for (_, v) in ma_list]) if ma_list else np.array([])

    if sar_coefs is None or len(sar_coefs) == 0:
        sar_list = []
        for name, val in model_fit.params.items():
            # seasonal AR patterns might include 'seasonal_ar.L1' or 'ar.S.L52' or 'ar.S.L1'
            if re.search(r'(seasonal.*ar|ar.*\.S\.L|ar\.S\.L)', name, re.I) or re.search(r'seasonal_ar', name, re.I):
                # try capture last L#
                mch = re.findall(r'L(\d+)$', name)
                idx = int(mch[0]) if mch else len(sar_list)+1
                sar_list.append((idx, val))
        sar_list.sort(key=lambda x: x[0])
        sar_coefs = np.array([v for (_, v) in sar_list]) if sar_list else np.array([])

    if sma_coefs is None or len(sma_coefs) == 0:
        sma_list = []
        for name, val in model_fit.params.items():
            if re.search(r'(seasonal.*ma|ma.*\.S\.L|ma\.S\.L)', name, re.I) or re.search(r'seasonal_ma', name, re.I):
                mch = re.findall(r'L(\d+)$', name)
                idx = int(mch[0]) if mch else len(sma_list)+1
                sma_list.append((idx, val))
        sma_list.sort(key=lambda x: x[0])
        sma_coefs = np.array([v for (_, v) in sma_list]) if sma_list else np.array([])
```

```
114
115     # Build left hand: differencing operators
116     left = ""
117     if D and D > 0:
118         left += r"\nabla^{" + str(D) + "} "
119     if d and d > 0:
120         left += r"\Delta^{" + str(d) + "} "
121     left += f"{var}_{{{time_idx}}}"
122
123     terms = []
124
125     # non-seasonal AR terms: phi_1 Z_{t-1} + ...
126     for i in range(min(p, len(ar_coefs))):
127         phi = ar_coefs[i]
128         sign = "-" if phi < 0 else "+"  # we will write + phi Z_{t-i}
129         coef = format_coef(abs(phi))
130         lag = i+1
131         terms.append(f"{sign}\\phi_{{{lag}}} {var}_{{{time_idx}-{lag}}}" if
    coef == "1.0000" else f"{sign}{coef} {var}_{{{time_idx}-{lag}}}")
132
133     # non-seasonal MA terms: - theta_1 eps_{t-1} ...
134     for i in range(min(q, len(ma_coefs))):
135         theta = ma_coefs[i]
136         sign = "-" if theta < 0 else "+"  # we will include sign and the
    minus from formula separately below
137         coef = format_coef(abs(theta))
138         lag = i+1
139         # in equation the MA part is "- theta_1 eps_{t-1}" (i.e. subtract MA
     terms)
140         # we'll represent as e_t - (theta1 e_{t-1} + ...)
141         terms.append(f"{sign}\\theta_{{{lag}}} \\varepsilon_{{{time_idx}-{
    lag}}}" if coef == "1.0000" else f"{sign}{coef} \\varepsilon_{{{time_idx
    }-{lag}}}")
142
143     # seasonal AR: + Phi_1 Z_{t-1S} ...
144     for i in range(min(P, len(sar_coefs))):
145         Phi = sar_coefs[i]
146         sign = "-" if Phi < 0 else "+"
147         coef = format_coef(abs(Phi))
148         lagS = (i+1) * m
149         terms.append(f"{sign}\\Phi_{{{i+1}}} {var}_{{{time_idx}-{lagS}}}" if
     coef == "1.0000" else f"{sign}{coef} {var}_{{{time_idx}-{lagS}}}")
150
151     # seasonal MA: - Theta_1 eps_{t-1S} ...
152     for i in range(min(Q, len(sma_coefs))):
153         Theta = sma_coefs[i]
154         sign = "-" if Theta < 0 else "+"
155         coef = format_coef(abs(Theta))
156         lagS = (i+1) * m
```

```
157         terms.append(f"{sign}\\Theta_{{{i+1}}} \\varepsilon_{{{time_idx}-{lagS}}}" if coef == "1.0000" else f"{sign}{coef} \\varepsilon_{{{time_idx}-{lagS}}}")
158 
159     # Build the RHS: start with epsilon_t (residuo)
160     rhs = "\\varepsilon_{" + time_idx + "}"
161 
162     # Append terms with signs. Note: the mathematical model you wrote has plus/minus placements:
163     # I'll append terms keeping the signs built above; then you can interpret +/-
164     if terms:
165         # join ensuring spacing
166         rhs += " " + " ".join(terms)
167 
168     # Compose full LaTeX equation environment
169     latex_eq = r"\begin{equation}" + "\n" \
170             + r"\nabla^{" + str(D) + r"} \Delta^{" + str(d) + r"} " + f"{var}_{{{time_idx}}} = " + rhs + "\n" \
171             + r"\end{equation}"
172 
173     # Post-process: clean up sequences like "+ -" or "+ +" and normalize signs:
174     latex_eq = latex_eq.replace("+ -", "- ").replace("- +", "- ").replace("  ", " ")
175 
176     return latex_eq
177 
178 
179 # Lista para armazenar metricas por produto
180 metricas = []
181 # DataFrame para salvar resultados, alertas e parametros
182 df_diagnostico = pd.DataFrame(columns=[
183     "Produto",
184     "order_p", "order_d", "order_q",
185     "seasonal_P", "seasonal_D", "seasonal_Q", "seasonal_m",
186     "LjungBox_p", "Normalidade_p", "ADF_p"
187 ])
188 
189 for produto in produtos:
190     print(f"Treinando modelo para produto: {produto}")
191 
192     train_cat = df_train[df_train["product_id"] == produto].set_index("ANO_SEMANA")["QTD_PRODUTOS"].sort_index()
193     test_cat = df_test[df_test["product_id"] == produto].set_index("ANO_SEMANA")["QTD_PRODUTOS"].sort_index()
194 
195     auto_model = auto_arima(
196         train_cat,
```

```
197         seasonal=True,
198         m=52,
199         max_p=3, max_q=3,
200         max_P=2, max_Q=2,
201         max_d=3, max_D=3,
202         stepwise=True,
203         suppress_warnings=True,
204         trace=False
205     )
206 
207     order = auto_model.order           # (p,d,q)
208     seasonal_order = auto_model.seasonal_order  # (P,D,Q,m)
209 
210     model = SARIMAX(train_cat, order=order, seasonal_order=seasonal_order,
    enforce_stationarity=False, enforce_invertibility=False)
211     model_fit = model.fit(disp=False)
212     latex_str = build_sarima_latex(model_fit, order, seasonal_order, var="Z"
    , time_idx="t")
213     # voce pode salvar no df_diagnostico
214     # df_diagnostico.loc[df_diagnostico["Produto"] == produto, "
    equacao_latex"] = latex_str
215     # ou apenas imprimir
216     # print(latex_str)
217     residuos = model_fit.resid
218 
219     # Testes
220     from statsmodels.stats.diagnostic import acorr_ljungbox
221     from scipy.stats import normaltest
222     from statsmodels.tsa.stattools import adfuller
223 
224     ljungbox_p = acorr_ljungbox(residuos, lags=[10], return_df=True)['
    lb_pvalue'].values[0]
225     # normal_p = normaltest(residuos)[1]
226     normal_p = jarque_bera(residuos)[1]
227     adf_p = adfuller(train_cat)[1]
228 
229     if ljungbox_p < 0.05:
230         print(f"[ALERTA] Residuos com autocorrelacao para produto {produto}
    (p={ljungbox_p:.4f})")
231 
232     if normal_p < 0.05:
233         print(f"[ALERTA] Residuos nao normais para produto {produto} (p={
    normal_p:.4f})")
234 
235     if adf_p > 0.05:
236         print(f"[ALERTA] Serie nao estacionaria para produto {produto} (ADF
    p={adf_p:.4f})")
237 
238     # Salvar no DataFrame
```

```
    df_diagnostico = pd.concat([
        df_diagnostico,
        pd.DataFrame([{
            "Produto": produto,
            "order_p": order[0],
            "order_d": order[1],
            "order_q": order[2],
            "seasonal_P": seasonal_order[0],
            "seasonal_D": seasonal_order[1],
            "seasonal_Q": seasonal_order[2],
            "seasonal_m": seasonal_order[3],
            "LjungBox_p": ljungbox_p,
            "Normalidade_p": normal_p,
            "ADF_p": adf_p,
            "equacao_latex": latex_str
        }])
    ], ignore_index=True)

    # Previsao para o horizonte de teste com intervalos
    forecast_res = model_fit.get_forecast(steps=len(test_cat))
    previsoes = forecast_res.predicted_mean
    conf_int = forecast_res.conf_int(alpha=0.05)  # 95% de confianca (padrao)

    # Arredondar e forcar minimo zero
    previsoes = np.round(previsoes).clip(lower=0)
    conf_int = np.round(conf_int).clip(lower=0)

    # Armazenar previsoes no DataFrame de teste
    df_test.loc[df_test['product_id'] == produto, 'Previsto_SARIMA'] = previsoes.values
    df_test.loc[df_test['product_id'] == produto, 'SARIMA_IC_Lower'] = conf_int.iloc[:, 0].values
    df_test.loc[df_test['product_id'] == produto, 'SARIMA_IC_Upper'] = conf_int.iloc[:, 1].values

    # Calcular metricas
    y_test = test_cat.values
    mae = mean_absolute_error(y_test, previsoes)
    rmse = mean_squared_error(y_test, previsoes, squared=False)
    smape_val = 100 * np.mean(2 * np.abs(previsoes - y_test) / (np.abs(y_test) + np.abs(previsoes)))

    metricas.append({
        'product_id': produto,
        'MAE': round(mae, 2),
        'RMSE': round(rmse, 2),
        'SMAPE': round(smape_val, 2)
    })
```

```
283
284 # Visualizar resultados:
285 print(df_diagnostico)
286
287
288 import jinja2
289 df_diagnostico['Parametros'] = df_diagnostico.apply(
290     lambda row: f"({row['order_p']},{row['order_d']},{row['order_q']})({row
    ['seasonal_P']},{row['seasonal_D']},{row['seasonal_Q']},{int(row['
    seasonal_m'])})",
291     axis=1
292 )
293
294 df_tabela = df_diagnostico[[
295     'Produto',
296     'Parametros',
297     'LjungBox_p',
298     'Normalidade_p',
299     'ADF_p'
300 ]].copy()
301
302 df_tabela['LjungBox_p'] = df_tabela['LjungBox_p'].map(lambda x: f"{x:.4f}")
303 df_tabela['Normalidade_p'] = df_tabela['Normalidade_p'].map(lambda x: f"{x
    :.4f}")
304 df_tabela['ADF_p'] = df_tabela['ADF_p'].map(lambda x: f"{x:.4f}")
305
306 latex_table = df_tabela.to_latex(
307     index=False,
308     caption="Diagnostico dos Modelos SARIMA para os Produtos",
309     label="tab:diagnostico_sarima",
310     float_format="%.4f",
311     column_format="c c c c c",
312     escape=False
313 )
314
315 print(latex_table)
316
317
318 df_test.to_csv("dados/resultados_sarima.csv", index=False)
```

B. Código Regressao Ridge

Apêndice B.1: Codigo para implementacao do modelo de Regressao ridge.

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

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

from sklearn.linear_model import Ridge

from sklearn.model_selection import ParameterSampler
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
    mean_absolute_percentage_error
from scipy.stats import randint as sp_randint, uniform as sp_uniform


def smape(y_true, y_pred):
    numerator = np.abs(y_true - y_pred)
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2
    # Evita divisao por zero, substitui denominador zero por 1 (ou pequeno
    valor)
    denominator = np.where(denominator == 0, 1, denominator)
    return np.mean(numerator / denominator) * 100


train = pd.read_csv('dados/train_filtered.csv')
test = pd.read_csv('dados/test_filtered.csv')


from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd
categorical_cols = ["SEMANA_ANO", "ANO", "MES", "TRIMESTRE", "product_id", "
    product_category_name"]

```

```
35
36 X = pd.get_dummies(train, columns=categorical_cols, prefix=categorical_cols)
       .copy()
37 X = X.astype(float)
38 vif_data = pd.DataFrame()
39 vif_data["feature"] = X.columns
40 vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
       shape[1])]
41
42 print(vif_data.sort_values("VIF", ascending=False))
43
44
45 produtos = test['product_id'].unique()
46 # Distribuicao para amostrar alphas(lambda)
47 param_dist = {'alpha': uniform(0.001, 100)}  # sorteia valores no intervalo
       [0.001, 100]
48
49 # Numero de amostras (experimentos de random search)
50 n_iter = 100
51 param_sampler = list(ParameterSampler(param_dist, n_iter=n_iter,
       random_state=42))
52
53 # Lista para armazenar metricas por produto
54 metricas = []
55
56 train = train.copy()
57 test = test.copy()
58
59 for produto in produtos:
60     train_prod = train[train['product_id'] == produto]
61     test_prod = test[test['product_id'] == produto]
62
63     if len(train_prod) < 2 or len(test_prod) == 0:
64         continue
65
66     X_train = pd.get_dummies(
67         train_prod.drop(columns=['QTD_PRODUTOS']),
68         columns=categorical_cols,
69         drop_first=False
70     )
71
72     y_train = train_prod['QTD_PRODUTOS']
73
74     X_test = pd.get_dummies(
75         test_prod.drop(columns=['QTD_PRODUTOS', 'previsao_linear'], errors='
       ignore'),
76         columns=categorical_cols,
77         drop_first=False
78     )
```

```
    y_test = test_prod['QTD_PRODUTOS']
    X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

    melhor_rmse = float("inf")
    melhor_modelo = None
    melhor_alpha = None
    melhor_previsoes = None

    # Random search: testar varios valores de alpha
    for params in param_sampler:
        modelo = Ridge(alpha=params['alpha'])
        modelo.fit(X_train, y_train)

        previsoes = np.maximum(modelo.predict(X_test).round(), 0)
        rmse = mean_squared_error(y_test, previsoes, squared=False)

        if rmse < melhor_rmse:
            melhor_rmse = rmse
            melhor_modelo = modelo
            melhor_alpha = params['alpha']
            melhor_previsoes = previsoes

    # Armazenar previsoes do melhor modelo
    test.loc[test['product_id'] == produto, 'previsao_linear'] =
   melhor_previsoes

    # Calcular metricas
    mae = mean_absolute_error(y_test, melhor_previsoes)
    rmse = mean_squared_error(y_test, melhor_previsoes, squared=False)
    mape = mean_absolute_percentage_error(y_test, melhor_previsoes)
    smape_val = smape(y_test.values, melhor_previsoes)

    metricas.append({
        'product_id': produto,
        'Alpha': round(melhor_alpha, 4),
        'MAE': round(mae, 2),
        'RMSE': round(rmse, 2),
        'MAPE': round(mape * 100, 2),
        'SMAPE': round(smape_val, 2),
        'Coeficientes': {col: round(coef, 4) for col, coef in zip(X_train.
   columns, melhor_modelo.coef_)},
        'Intercepto': round(melhor_modelo.intercept_, 4)
    })

# DataFrame com metricas
df_metricas = pd.DataFrame(metricas)
print(df_metricas.sort_values('product_id'))
```

```
126 df_test.to_csv("dados/resultados_ridge.csv", index=False)
```

C. Código LightGBM(LGBM)

Apêndice C.1: Codigo para implementacao do modelo LightGBM (LGBM).

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

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

from sklearn.linear_model import Ridge

from sklearn.model_selection import ParameterSampler
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
    mean_absolute_percentage_error
from scipy.stats import randint as sp_randint, uniform as sp_uniform


def smape(y_true, y_pred):
    numerator = np.abs(y_true - y_pred)
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2
    # Evita divisao por zero, substitui denominador zero por 1 (ou pequeno
    valor)
    denominator = np.where(denominator == 0, 1, denominator)
    return np.mean(numerator / denominator) * 100


train = pd.read_csv('dados/train_filtered.csv')
test = pd.read_csv('dados/test_filtered.csv')


param_dist = {
    'num_leaves': sp_randint(5, 31),
    'max_depth': sp_randint(3, 10),
    'learning_rate': sp_uniform(0.01, 0.3),
    'n_estimators': sp_randint(50, 200),
```

```
    'min_child_samples': sp_randint(5, 30)
    # ,'bagging_fraction': sp_uniform(0.5, 0.5),  # valores tipicos entre 0.5 e 1
    # 'bagging_freq': sp_randint(1, 10)
}

metricas_lgbm = []

for produto in produtos:
    train_prod = train[train['product_id'] == produto]
    test_prod = test[test['product_id'] == produto]

    X_train = train_prod.drop(columns=['QTD_PRODUTOS'])
    y_train = train_prod['QTD_PRODUTOS']

    X_test = test_prod.drop(columns=['QTD_PRODUTOS', 'previsao_lgbm', 'previsao_linear'], errors='ignore')
    y_test = test_prod['QTD_PRODUTOS']

    modelo_base = LGBMRegressor()

    random_search = RandomizedSearchCV(
        estimator=modelo_base,
        param_distributions=param_dist,
        n_iter=20,
        scoring='neg_mean_absolute_error',
        cv=3,
        verbose=0,
        n_jobs=-1,
        random_state=42
    )

    random_search.fit(X_train, y_train)

    melhor_modelo = random_search.best_estimator_

    previsoes = np.maximum(melhor_modelo.predict(X_test).round(), 0)

    test.loc[test['product_id'] == produto, 'previsao_lgbm'] = previsoes

    mae = mean_absolute_error(y_test, previsoes)
    rmse = mean_squared_error(y_test, previsoes, squared=False)
    mape = mean_absolute_percentage_error(y_test, previsoes)
    smape_val = smape(y_test.values, previsoes)

    # Salvar metricas e hiperparametros usados
    metricas_lgbm.append({
        'product_id': produto,
        'MAE': round(mae, 2),
```

```
        'RMSE': round(rmse, 2),
        'MAPE': round(mape * 100, 2),
        'SMAPE': round(smape_val, 2),
        'num_leaves': melhor_modelo.num_leaves,
        'max_depth': melhor_modelo.max_depth,
        'learning_rate': round(melhor_modelo.learning_rate, 4),
        'n_estimators': melhor_modelo.n_estimators,
        'min_child_samples': melhor_modelo.min_child_samples
    })

df_metricas_lgbm = pd.DataFrame(metricas_lgbm)
print(df_metricas_lgbm.sort_values('product_id'))

df_test.to_csv("dados/resultados_LGBM.csv", index=False)
```

D. CODIGO REDE NEURAL(LSTM)

Apêndice D.1: Codigo para implementacao da rede neural recorrente(LSTM).

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import itertools
import random

from sklearn.preprocessing import MinMaxScaler
import torch.optim.lr_scheduler as lr_scheduler
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.metrics import mean_absolute_error, mean_squared_error,
    mean_absolute_percentage_error, r2_score


df_train = pd.read_csv('dados/train_filtered.csv')
df_test_orig = pd.read_csv('dados/test_filtered.csv')


seq_length = 1  # Numero de semanas usadas para prever a proxima

# Criando uma copia do conjunto de treino para modificar
df_train_adj = df_train.copy()


# Movendo os ultimos seq_length registros de cada product_id de train para
    test
extra_rows = []
for product in df_train_adj['product_id'].unique():
    product_rows = df_train_adj[df_train_adj['product_id'] == product]
    last_seq_rows = product_rows.iloc[-seq_length:]
    extra_rows.append(last_seq_rows)

extra_rows = pd.concat(extra_rows)
df_test = pd.concat([extra_rows, df_test_orig]).reset_index(drop=True)
df_train_adj = df_train_adj[~df_train_adj.index.isin(extra_rows.index)]

```

```
36
37 scalers_X, scalers_y = {}, {}
38
39 def normalize_data(df, target_column):
40     X_scaled_dict, y_scaled_dict = {}, {}
41
42     for product in df['product_id'].unique():
43         product_data = df[df['product_id'] == product]
44         X_product = product_data.drop(columns=[target_column, 'product_id'])
    .values
45         y_product = product_data[[target_column]].values
46
47         # VALIDAR QUAIS FEATURES ESTaO ENTRANDO
48         print(f"\n[Produto {product}] Features de entrada (colunas):")
49         print(product_data.drop(columns=[target_column, 'product_id']).
    columns.tolist())
50         print(f"[Produto {product}] Primeira linha das features (valores):")
51         print(X_product[0])
52
53         scaler_X = MinMaxScaler()
54         scaler_y = MinMaxScaler()
55
56         X_scaled_dict[product] = scaler_X.fit_transform(X_product)
57         y_scaled_dict[product] = scaler_y.fit_transform(y_product)
58
59         scalers_X[product] = scaler_X
60         scalers_y[product] = scaler_y
61
62     return X_scaled_dict, y_scaled_dict
63
64
65 # Normalizando os dados sem misturar produtos
66 target_column = "QTD_PRODUTOS"
67 X_train_scaled, y_train_scaled = normalize_data(df_train_adj, target_column)
68 X_test_scaled, y_test_scaled = normalize_data(df_test, target_column)
69
70
71 def create_sequences_per_product(df, X_scaled_dict, y_scaled_dict,
    seq_length):
72     Xs, ys = [], []
73
74     for product in df['product_id'].unique():
75         X_product = X_scaled_dict[product]  # Obtendo os dados normalizados
    do produto
76         y_product = y_scaled_dict[product]  # Obtendo a variavel alvo do
    produto
77
78         for i in range(len(X_product) - seq_length):
79             Xs.append(X_product[i : i + seq_length])
```

```
            ys.append(y_product[i + seq_length])

    return np.array(Xs), np.array(ys)

# Criando sequencias corretamente
X_train_seq, y_train_seq = create_sequences_per_product(df_train_adj, X_train_scaled, y_train_scaled, seq_length)
X_test_seq, y_test_seq = create_sequences_per_product(df_test, X_test_scaled, y_test_scaled, seq_length)


categorical_cols = ['SEMANA_ANO', 'MES', 'TRIMESTRE', 'ANO']
for product in df_train_adj['product_id'].unique():
    # Colunas originais para esse produto (antes de dummyficar)
    feature_names_train = df_train_adj[df_train_adj['product_id'] == product].drop(
        columns=['product_id', 'QTD_PRODUTOS']
    ).columns.tolist()

    feature_names_test = df_test[df_test['product_id'] == product].drop(
        columns=['product_id', 'QTD_PRODUTOS']
    ).columns.tolist()

    # Unir para definir todas as colunas dummy
    combined = pd.concat([
        pd.DataFrame(X_train_scaled[product], columns=feature_names_train),
        pd.DataFrame(X_test_scaled[product], columns=feature_names_test)
    ], axis=0)

    combined = pd.get_dummies(combined, columns=categorical_cols, drop_first=False)

    # Lista final de colunas dummy que sera usada como referencia
    all_columns = combined.columns.tolist()

def create_dummies(df, categorical_cols, all_columns=None):
    """
    df: DataFrame original
    categorical_cols: colunas categoricas a dummyficar
    all_columns: lista de todas as colunas finais (treino+teste)
    """
    df_copy = df.copy()
    df_copy = pd.get_dummies(df_copy, columns=categorical_cols, drop_first=False)

    if all_columns is not None:
        # Adiciona colunas que faltam como zero
        for col in all_columns:
            if col not in df_copy.columns:
```

```
                df_copy[col] = 0
        # Reordenar
        df_copy = df_copy[all_columns]

    return df_copy.values


# Convertendo para tensores
X_train_tensor = torch.tensor(X_train_seq, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train_seq, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_seq, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test_seq, dtype=torch.float32)
# Criando DataLoader
batch_size = 32
train_loader = DataLoader(TensorDataset(X_train_tensor, y_train_tensor),
    batch_size=batch_size, shuffle=True)
test_loader = DataLoader(TensorDataset(X_test_tensor, y_test_tensor),
    batch_size=batch_size, shuffle=False)


# Definicao do modelo LSTM
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size=128, num_layers=3,
    dropout_rate=0.2):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(
            input_size,
            hidden_size,
            num_layers,
            batch_first=True,
            dropout=dropout_rate if num_layers > 1 else 0
        )
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        last_time_step = lstm_out[:, -1, :]
        return self.fc(last_time_step)

# Adicionando Early Stopping
class EarlyStopping:
    def __init__(self, patience=30, min_delta=0.001):
        self.patience = patience
        self.min_delta = min_delta
        self.best_loss = float('inf')
        self.counter = 0

    def step(self, val_loss):
        if val_loss < self.best_loss - self.min_delta:
```

```
            self.best_loss = val_loss
            self.counter = 0
        else:
            self.counter += 1
            if self.counter >= self.patience:
                return True  # Parar treino
        return False


# Hiperparametros
input_size = 84
hidden_size = 128  # Reduzi de 64 para 32
num_layers = 3  # Deixei so 1 camada para simplificar
dropout_rate = 0.2  # Adicionei Dropout para evitar overfitting
learning_rate = 0.0001
num_epochs = 1000
patience_epochs = 30

# Inicializando modelo, funcao de perda e otimizador
model = LSTMModel(input_size, hidden_size, num_layers, dropout_rate)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)


categorical_cols = ['SEMANA_ANO', 'MES', 'TRIMESTRE', 'ANO']
param_grid = {
    'hidden_size': [16, 32, 64, 128],
    'num_layers': [1, 2, 3],
    'dropout': [0.0, 0.2, 0.3, 0.5],
    'learning_rate': [1e-4, 5e-4, 1e-3, 5e-3]
}

num_trials = 10
results = {}

def create_dummies(df, categorical_cols, all_columns=None):
    df_copy = df.copy()
    df_copy = pd.get_dummies(df_copy, columns=categorical_cols, drop_first=
   False)

    if all_columns is not None:
        # Adiciona colunas que nao existem como zeros
        for col in all_columns:
            if col not in df_copy.columns:
                df_copy[col] = 0
        # Reordena para manter consistencia
        df_copy = df_copy[all_columns]

    return df_copy.values
```

```
218
219 for product in df_train_adj['product_id'].unique():
220     print(f"\nOtimizando modelo para o produto {product}...")
221
222     # --- Feature names originais ---
223     feature_names_train = df_train_adj[df_train_adj['product_id'] == product
    ].drop(
224         columns=['product_id', 'QTD_PRODUTOS']
225     ).columns.tolist()
226
227     feature_names_test = df_test[df_test['product_id'] == product].drop(
228         columns=['product_id', 'QTD_PRODUTOS']
229     ).columns.tolist()
230
231     # --- Criar DataFrames ---
232     X_train_df = pd.DataFrame(X_train_scaled[product], columns=
    feature_names_train)
233     X_test_df = pd.DataFrame(X_test_scaled[product], columns=
    feature_names_test)
234
235     # --- Dummyficar temporariamente para definir todas as colunas ---
236     combined = pd.concat([X_train_df, X_test_df], axis=0)
237     combined = pd.get_dummies(combined, columns=categorical_cols, drop_first
    =False)
238     all_columns = combined.columns.tolist()
239
240     # --- Dummyficar treino e teste usando todas as colunas ---
241     X_product = create_dummies(X_train_df, categorical_cols, all_columns)
242     X_test_product = create_dummies(X_test_df, categorical_cols, all_columns
    )
243     y_product = y_train_scaled[product]
244     y_test_product = y_test_scaled.get(product)
245
246     # --- Ajustar input_size para LSTM ---
247     input_size = X_product.shape[1]
248
249     # --- Criar sequencias para treino ---
250     Xs, ys = [], []
251     for i in range(len(X_product) - seq_length):
252         Xs.append(X_product[i:i+seq_length])
253         ys.append(y_product[i+seq_length])
254     if len(Xs) == 0:
255         print(f"Produto {product} ignorado por falta de dados.")
256         continue
257     X_tensor = torch.tensor(Xs, dtype=torch.float32)
258     y_tensor = torch.tensor(ys, dtype=torch.float32)
259
260     # --- Divisao treino/validacao ---
261     val_size = int(0.2 * len(X_tensor))
```

```
    train_size = len(X_tensor) - val_size
    train_dataset, val_dataset = random_split(TensorDataset(X_tensor, y_tensor), [train_size, val_size])
    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

    best_val_loss = float('inf')
    best_model = None
    best_params = None

    # --- Random Search ---
    for trial in range(num_trials):
        params = {
            'hidden_size': random.choice(param_grid['hidden_size']),
            'num_layers': random.choice(param_grid['num_layers']),
            'dropout': random.choice(param_grid['dropout']),
            'learning_rate': random.choice(param_grid['learning_rate'])
        }

        model = LSTMModel(input_size, params['hidden_size'], params['num_layers'], params['dropout'])
        optimizer = torch.optim.Adam(model.parameters(), lr=params['learning_rate'])
        criterion = nn.MSELoss()
        scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=10, verbose=False)
        early_stopping = EarlyStopping(patience=20)

        for epoch in range(num_epochs):
            model.train()
            for X_batch, y_batch in train_loader:
                optimizer.zero_grad()
                y_pred = model(X_batch)
                loss = criterion(y_pred, y_batch)
                loss.backward()
                optimizer.step()

            # Validacao
            model.eval()
            val_loss = 0
            with torch.no_grad():
                for X_val, y_val in val_loader:
                    y_pred = model(X_val)
                    val_loss += criterion(y_pred, y_val).item()
            val_loss /= len(val_loader)
            scheduler.step(val_loss)
            if early_stopping.step(val_loss):
                break

```

```
307         if val_loss < best_val_loss:
308             best_val_loss = val_loss
309             best_model = model
310             best_params = params
311 
312     print(f"Melhor combinacao para produto {product}: {best_params}, Val
    Loss: {best_val_loss:.4f}")
313 
314     # --- Criar sequencias para teste ---
315     if X_test_product is None or y_test_product is None or len(
    X_test_product) <= seq_length:
316         print(f"Produto {product} ignorado no teste por falta de dados.")
317         continue
318 
319     Xs_test, ys_test = [], []
320     for i in range(len(X_test_product) - seq_length):
321         Xs_test.append(X_test_product[i:i+seq_length])
322         ys_test.append(y_test_product[i+seq_length])
323     X_test_tensor = torch.tensor(Xs_test, dtype=torch.float32)
324     y_test_tensor = torch.tensor(ys_test, dtype=torch.float32)
325     test_loader = DataLoader(TensorDataset(X_test_tensor, y_test_tensor),
    batch_size=32, shuffle=False)
326 
327     best_model.eval()
328     preds = []
329     with torch.no_grad():
330         for X_batch, _ in test_loader:
331             y_pred = best_model(X_batch)
332             preds.append(y_pred.numpy())
333     preds = np.concatenate(preds)
334 
335     # --- Metricas ---
336     scaler_y = scalers_y[product]
337     preds_orig = scaler_y.inverse_transform(preds)
338     y_true_orig = scaler_y.inverse_transform(y_test_tensor.numpy())
339     y_true_orig = np.round(y_true_orig)
340 
341     mae = mean_absolute_error(y_true_orig, preds_orig)
342     rmse = mean_squared_error(y_true_orig, preds_orig, squared=False)
343     mape = np.mean(np.abs((y_true_orig - preds_orig) / y_true_orig)) * 100
344     smape_val = np.mean(2 * np.abs(preds_orig - y_true_orig) / (np.abs(
    y_true_orig) + np.abs(preds_orig))) * 100
345 
346     print(f"Produto {product} - MAE: {mae:.4f}, RMSE: {rmse:.4f}, MAPE: {
    mape:.2f}%, SMAPE: {smape_val:.2f}%")
347 
348     results[product] = {
349         'model': best_model,
350         'preds': preds_orig.flatten(),
```

```
        'y_true': y_true_orig.flatten(),
        'mae': mae,
        'rmse': rmse,
        'mape': mape,
        'smape': smape_val,
        'best_params': best_params
    }


list_dfs = []
for product, data in results.items():
    # Filtrar df_test_orig so para o produto atual
    df_test_prod = df_test_orig[df_test_orig['product_id'] == product].
    reset_index(drop=True)

    # Ajustar datas considerando o seq_length (previsoes so a partir dai)
    ano_semana = (df_test_prod['ANO'].astype(str) + '_' + df_test_prod['
    SEMANA_ANO'].astype(str)).reset_index(drop=True)

    # Os valores previstos e reais estao no formato correto, so garantir o
    tamanho correto
    y_true = df_test_prod['QTD_PRODUTOS']
    preds = data['preds']

    # Criar DataFrame para o produto
    df_prod = pd.DataFrame({
        "ANO_SEMANA": ano_semana,
        "Produto": product,
        "Real": y_true,
        "Previsto_LSTM": np.round(preds)
    })

    list_dfs.append(df_prod)

# Concatenar todos os produtos
df_resultados = pd.concat(list_dfs).reset_index(drop=True)

# Salvando em CSV
df_resultados.to_csv("dados/resultados_lstm.csv", index=False)
```

E. Código Resultados Modelos

Apêndice E.1: Codigo para resultado dos modelos.

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

def smape(y_true, y_pred):
    numerator = np.abs(y_true - y_pred)
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2
    # Evita divisao por zero, substitui denominador zero por 1 (ou pequeno valor)
    denominator = np.where(denominator == 0, 1, denominator)
    return np.mean(numerator / denominator) * 100


df_lstm = pd.read_csv('dados/resultados_lstm.csv')
df_ridge = pd.read_csv('dados/resultados_ridge.csv')
df_lgbm = pd.read_csv('dados/resultados_LGBM.csv')
df_sarima = pd.read_csv('dados/resultados_sarima.csv')


df_sarima = df_sarima.rename(columns={
    'product_id': 'Produto',
    'QTD_PRODUTOS': 'Real',
    'SARIMA_IC_Lower': 'LI_SARIMA',
    'SARIMA_IC_Upper': 'LS_SARIMA'
})

merged_df = pd.merge(df_lstm, df_ridge, on=['ANO_SEMANA', 'Produto'], how='outer')
merged_df = pd.merge(merged_df, df_lgbm, on=['ANO_SEMANA', 'Produto'], how='outer')
merged_df = pd.merge(merged_df, df_sarima, on=['ANO_SEMANA', 'Produto'], how='outer')

df = merged_df[['ANO_SEMANA', 'Produto', 'Real', 'Previsto_LSTM', 'Previsto_reg', 'Previsto_lgbm', 'Previsto_SARIMA', 'LI_SARIMA', 'LS_SARIMA']]
df = df.drop_duplicates(subset=['ANO_SEMANA', 'Produto'], keep='first').
```

```
    reset_index(drop=True)
32
33
34 # Substituir NaN por 0
35 df = df.fillna(0)
36
37 # Lista para salvar as metricas por Produto
38 metrics_table = []
39
40
41 # Agrupar por Produto e calcular metricas para cada modelo de previsao
42 for Produto, group in df.groupby('Produto'):
43     true_values = group['Real']
44
45     # Calculo das metricas para LSTM
46     mae_lstm = mean_absolute_error(true_values, group['Previsto_LSTM'])
47     rmse_lstm = np.sqrt(mean_squared_error(true_values, group['Previsto_LSTM
    ']))
48     smape_lstm = smape(true_values, group['Previsto_LSTM'])
49
50     # Calculo das metricas para Regressao
51     mae_reg = mean_absolute_error(true_values, group['Previsto_reg'])
52     rmse_reg = np.sqrt(mean_squared_error(true_values, group['Previsto_reg'
    ]))
53     smape_reg = smape(true_values, group['Previsto_reg'])
54
55     # Calculo das metricas para LGBM
56     mae_lgbm = mean_absolute_error(true_values, group['Previsto_lgbm'])
57     rmse_lgbm = np.sqrt(mean_squared_error(true_values, group['Previsto_lgbm
    ']))
58     smape_lgbm = smape(true_values, group['Previsto_lgbm'])
59
60     # Calculo das metricas para SARIMA
61     mae_sarima = mean_absolute_error(true_values, group['Previsto_SARIMA'])
62     rmse_sarima = np.sqrt(mean_squared_error(true_values, group['
    Previsto_SARIMA']))
63     smape_sarima = smape(true_values, group['Previsto_SARIMA'])
64
65     # Salvar as metricas para o Produto em um dicionario
66     metrics_table.append({
67         'Produto': Produto,
68         'MAE_LSTM': round(mae_lstm, 2), 'RMSE_LSTM': round(rmse_lstm, 2), '
    SMAPE_LSTM': round(smape_lstm, 2),
69         'MAE_Reg': round(mae_reg, 2), 'RMSE_Reg': round(rmse_reg, 2), '
    SMAPE_Reg': round(smape_reg, 2),
70         'MAE_LGBM': round(mae_lgbm, 2), 'RMSE_LGBM': round(rmse_lgbm, 2), '
    SMAPE_LGBM': round(smape_lgbm, 2),
71         'MAE_SARIMA': round(mae_sarima, 2), 'RMSE_SARIMA': round(rmse_sarima
    , 2), 'SMAPE_SARIMA': round(smape_sarima, 2),
```

```
    })

# Criar um DataFrame com as metricas
metrics_df = pd.DataFrame(metrics_table)


import pandas as pd

# Supondo que metrics_df ja existe
# Lista de modelos e mapeamento das colunas
modelos = {
    "LSTM": ["MAE_LSTM", "RMSE_LSTM", "SMAPE_LSTM"],
    "Linear": ["MAE_Reg", "RMSE_Reg", "SMAPE_Reg"],
    "LightGBM": ["MAE_LGBM", "RMSE_LGBM", "SMAPE_LGBM"],
    "SARIMA": ["MAE_SARIMA", "RMSE_SARIMA", "SMAPE_SARIMA"],
}

# Reformata para tabela longa
tabelas = []
for modelo, cols in modelos.items():
    temp = metrics_df[["Produto"] + cols].copy()
    temp["Produto"] = temp["Produto"] + 1
    temp["Modelo"] = modelo
    temp = temp.rename(columns={
        cols[0]: "MAE",
        cols[1]: "RMSE",
        cols[2]: "sMAPE"
    })
    tabelas.append(temp)

metrics_long = pd.concat(tabelas, ignore_index=True)

# Reordena colunas
metrics_long = metrics_long[["Produto", "Modelo", "MAE", "RMSE", "sMAPE"]]

# Ordena por Produto e Modelo na ordem desejada
order_modelos = ["LightGBM", "LSTM", "Linear", "SARIMA"]
metrics_long["Modelo"] = pd.Categorical(metrics_long["Modelo"], categories=
    order_modelos, ordered=True)
metrics_long = metrics_long.sort_values(["Produto", "Modelo"]).reset_index(
    drop=True)

# Tabela de produtos 1 a 5
subset1 = metrics_long[metrics_long["Produto"].between(0, 11)]
latex1 = subset1.to_latex(index=False, column_format="llllll", caption="
    desempenho dos modelos para os produtos 1 a 5", label="tab:metricas_1_5")

# Tabela de produtos 6 a 10
subset2 = metrics_long[metrics_long["Produto"].between(6, 11)]
```

```
latex2 = subset2.to_latex(index=False, column_format="lllll", caption="
    desempenho dos modelos para os produtos 6 a 10", label="tab:metricas_6_10
    ")

print(latex1)
# print(latex2)

import matplotlib.pyplot as plt
import os

output_dir = "graficos_produtos"
os.makedirs(output_dir, exist_ok=True)

# Criar graficos para cada Produto
for Produto, group in df.groupby('Produto'):
    # Ordenar os dados por ANO_SEMANA dentro do Produto
    group = group.sort_values(by=['ANO_SEMANA'])

    plt.figure(figsize=(10, 6))

    # Plotando a serie temporal
    plt.plot(group['ANO_SEMANA'], group['Real'], label="Real", color="blue",
     linewidth=2)
    plt.plot(group['ANO_SEMANA'], group['Previsto_LSTM'], label="LSTM",
    linestyle="dashed", color="red")
    plt.plot(group['ANO_SEMANA'], group['Previsto_reg'], label="Regressao",
    linestyle="dashed", color="green")
    plt.plot(group['ANO_SEMANA'], group['Previsto_lgbm'], label="LGBM",
    linestyle="dashed", color="orange")
    plt.plot(group['ANO_SEMANA'], group['Previsto_SARIMA'], label="SARIMA",
    linestyle="dashed", color="purple")

    # Adicionar intervalo de confianca SARIMA, se disponivel
    if 'LI_SARIMA' in group.columns and 'LS_SARIMA' in group.columns:
        plt.fill_between(
            group['ANO_SEMANA'],
            group['LI_SARIMA'],
            group['LS_SARIMA'],
            color='purple',
            alpha=0.2,
            label='Intervalo SARIMA'
        )

    # Configuracoes do grafico
    plt.title(f"Comparacao Real vs Previsto - Produto: {Produto + 1}",
    fontsize=18)
    plt.xlabel("ANO_SEMANA", fontsize=18)
    plt.ylabel("Volume de Vendas", fontsize=18)
    plt.xticks(fontsize=12, rotation=45)
```

```
159     plt.yticks(fontsize=12)
160     plt.legend()
161     plt.grid(True)
162 
163     # Nome do arquivo
164     filename = os.path.join(output_dir, f"comparacao_produto_{Produto+1}.png
        ")
165 
166     # Salvar figura
167     plt.savefig(filename, dpi=300, bbox_inches="tight")
168     plt.close()
```