UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Departamento de Estatística

Renato Godoi da Cruz

Geração de dados com Algoritmo Genético e aplicação de *Machine Learning* na otimização de estruturas metálicas

Belo Horizonte
2025

Renato Godoi da Cruz

Geração de dados com Algoritmo Genético e aplicação de *Machine Learning* na otimização de estruturas metálicas

Monografia de especialização apresentada ao Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial à obtenção do título de Especialista em Estatística.

Orientação: Uriel Moreira Silva

Belo Horizonte
2025

Cruz, Renato Godoi da.

C957g Geração de dados com algoritmo genético e aplicação de machine learning na otimização de estruturas metálicas [recurso eletrônico] / Renato Godoi da Cruz – 2025.
1 recurso online (55 f. il., color.) : pdf.

Orientador: Uriel Moreira Silva.

Monografia (especialização) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Estatística.
Referências: f. 41.

1. Estatística. 2. Métodos estatísticos - Estruturas metálicas. 3. Controle preditivo - Floresta Aleatória. 4. Aprendizado de computador. 5. Algoritmos genéticos. I. Silva, Uriel Moreira de. II. Universidade Federal de Minas Gerais, Instituto de Ciência Exatas, Departamento de Estatística. III. Título.

CDU 519.2(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEx

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Estatística
Programa de Pós-Graduação / Especialização
Av. Pres. Antônio Carlos, 6627 - Pampulha
31270-901 – Belo Horizonte – MG

E-mail: pgest@ufmg.br
Tel: 3409-5923 – FAX: 3409-5924

ATA DO 339ª. TRABALHO DE FIM DE CURSO DE ESPECIALIZAÇÃO EM ESTATÍSTICA DE RENATO GODOI DA CRUZ.

Aos quatorze dias do mês de fevereiro de 2025, às 17:00 horas, com utilização de recursos de videoconferência a distância, reuniram-se os professores abaixo relacionados, formando a Comissão Examinadora homologada pela Comissão do Curso de Especialização em Estatística, para julgar a apresentação do trabalho de fim de curso do aluno **Renato Godoi da Cruz**, intitulado: "*Geração de dados com algoritmo genético e aplicação de Machine Learning na otimização de estruturas metálicas"*, como requisito para obtenção do Grau de Especialista em Estatística. Abrindo a sessão, o Presidente da Comissão, Professor Uriel Moreira Silva – Orientador, após dar conhecimento aos presentes do teor das normas regulamentares, passou a palavra ao candidato para apresentação de seu trabalho. Seguiu-se a arguição pelos examinadores com a respectiva defesa do candidato. Após a defesa, os membros da banca examinadora reuniram-se sem a presença do candidato e do público, para julgamento e expedição do resultado final. Foi atribuída a seguinte indicação: o candidato foi considerado Aprovado condicional às modificações sugeridas pela banca examinadora no prazo de 30 dias a partir da data de hoje por unanimidade. O resultado final foi comunicado publicamente ao candidato pelo Presidente da Comissão. Nada mais havendo a tratar, o Presidente encerrou a reunião e lavrou a presente Ata, que será assinada por todos os membros participantes da banca examinadora. Belo Horizonte, 14 de fevereiro de 2025.

gov.br Documento assinado digitalmente
URIEL MOREIRA SILVA
Data: 01/03/2025 18:20:41-0300
Verifique em https://validar.iti.gov.br

Prof. Uriel Moreira Silva (orientador)
DEST/UFMG

gov.br Documento assinado digitalmente
CASSIUS HENRIQUE XAVIER OLIVEIRA
Data: 03/03/2025 20:01:32-0300
Verifique em https://validar.iti.gov.br

Prof. Cássius Henrique Xavier Oliveira
DEST/UFMG

AGRADECIMENTOS

Agradeço à minha mãe, Ofélia, por seu apoio incondicional, à Caroline pela paciência e companheirismo ao longo dos últimos anos, ao meu orientador, Uriel, que não hesitou em me ajudar nesta tarefa e me acompanhou até o último minuto, e ao professor Cássius, cuja participação na banca avaliadora e sugestões valiosas enriqueceram ainda mais este trabalho.

RESUMO

Este trabalho apresenta uma abordagem integrada para otimizar projetos de estruturas metálicas, combinando Algoritmos Genéticos, coleta de dados e *Machine Learning*. Os Algoritmos Genéticos foram empregados para gerar uma base robusta a partir de diferentes configurações estruturais, fornecendo parâmetros como características de perfis metálicos e posicionamento estrutural. Esses dados foram analisados estatisticamente e utilizados em modelos preditivos, incluindo *Random Forest*, *Gradient Boosting* e *XGBoost*. Os resultados indicam que os modelos baseados em *boosting* (*Gradient Boosting* e *XGBoost*) tiveram um desempenho superior ao *Random Forest* em todas as métricas analisadas. O *Gradient Boostin*g obteve um MAE de 4.7674, MSE de 53.6243 e RMSE de 7.3229, enquanto o *XGBoost* apresentou um MAE de 4.7255, MSE de 54.0651 e RMSE de 7.3529. Ambos os modelos atingiram valores semelhantes de R^2 Score (0.5844 para *Gradient Boosting* e 0.5809 para *XGBoost*), demonstrando alta capacidade de explicação da variância dos dados. O estudo destaca o potencial da integração entre Algoritmos Genéticos, análise estatística e *Machine Learning*, demonstrando que modelos baseados em *boosting* conseguem capturar melhor os padrões estruturais dos dados gerados neste trabalho. Os resultados obtidos sugerem que essa abordagem pode promover projetos mais eficientes, econômicos e sustentáveis na engenharia de estruturas metálicas.

Palavras-chave: otimização de estruturas metálicas; algoritmos genéticos; *machine learning*; *random forest*; *gradient boosting*; *xgboost*.

ABSTRACT

This work presents an integrated approach to optimizing metal structure design by combining genetic algorithms, data collection, and Machine Learning. Genetic Algorithms were employed to generate a robust dataset from various structural configurations, providing parameters such as metallic profile characteristics and structural positioning. These data were statistically analyzed and used in predictive models, including Random Forest, Gradient Boosting, and XGBoost. The results indicate that boosting-based models (Gradient Boosting and XGBoost) outperformed Random Forest across all evaluated metrics. Gradient Boosting achieved an MAE of 4.7674, MSE of 53.6243, and RMSE of 7.3229, while XGBoost reported an MAE of 4.7255, MSE of 54.0651, and RMSE of 7.3529. Both models reached similar R^2 scores (0.5844 for Gradient Boosting and 0.5809 for XGBoost), demonstrating a strong ability to explain data variance. This study highlights the potential of integrating genetic algorithms, statistical analysis, and Machine Learning, showing that boosting-based models can better capture the structural patterns in the data. The findings suggest that this approach can contribute to more efficient, cost-effective, and sustainable metal structure engineering projects.

Keywords: metallic structure optimization; genetic algorithms; machine learning; random forest; gradient boosting; xgboost.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A otimização de estruturas metálicas é um campo de grande relevância na Engenharia Civil, diretamente relacionado à eficiência, custo e segurança das construções. Tradicionalmente, esse processo baseia-se em abordagens empíricas e iterativas, nas quais diferentes configurações estruturais são testadas até se atingir um projeto satisfatório (Cruz, 2021). No entanto, essas metodologias convencionais frequentemente demandam tempo e recursos consideráveis, além de estarem sujeitas a limitações na exploração de soluções inovadoras. Um dos principais desafios para a aplicação de técnicas avançadas, como o *Machine Learning* (ML), é a escassez de bancos de dados robustos e diversificados, essenciais para o treinamento e validação de modelos preditivos (Thai, 2022).

Com o avanço das tecnologias computacionais, métodos inovadores, como Algoritmos Genéticos (GA, do inglês *Genetic Algorithms*) e *Machine Learning* têm ganhado destaque por sua capacidade de automatizar processos, analisar grandes volumes de dados e identificar soluções ótimas de forma mais eficiente (Holland, 1975; Mitchell, 1998). Os Algoritmos Genéticos, inspirados na evolução biológica, permitem explorar um amplo espaço de soluções por meio de operações como seleção, cruzamento e mutação, enquanto o *Machine Learning* possibilita a análise preditiva de dados complexos, identificando padrões e relações não-lineares (Goldberg, 1989; Goodfellow et al., 2016).

Neste contexto, este trabalho propõe uma abordagem metodológica para otimizar o projeto de estruturas metálicas, integrando Algoritmos Genéticos para a geração de dados e técnicas de *Machine Learning* para análise preditiva. A metodologia visa superar as limitações das abordagens tradicionais, oferecendo uma ferramenta eficiente para apoiar decisões no processo de projeto. A utilização de Algoritmos Genéticos permite gerar um conjunto diversificado de soluções estruturais, enquanto modelos de ML, como *Random Forest*, *Gradient Boosting* e *XGBoost*, são aplicados para prever o desempenho estrutural com base em métricas como erro absoluto médio (*MAE*, do inglês *Mean Absolute Error*), erro quadrático médio (*MSE*, do *inglês Mean Squared Error*), raiz do erro quadrático médio (*RMSE*, do inglês *Root Mean-Square Error*) e coeficiente de determinação (R^2, do inglês *R-squared*). Essa abordagem tem o potencial de identificar configurações mais econômicas, eficientes e sustentáveis, contribuindo para avanços na engenharia estrutural.

O estudo é dividido em quatro etapas principais:

I. Geração de dados: utiliza-se um Algoritmo Genético implementado em *Python*, integrado ao *Rhinoceros* e *Grasshopper*, aplicado com o objetivo de minimizar o peso da estrutura e gerar uma massa de dados;

II. Coleta de dados: utiliza-se um algoritmo em *Python* para coleta dos dados gerados na etapa anterior. Os dados incluem variáveis de entrada da geometria e dos perfis metálicos e a variável de saída, ou seja, o peso total da estrutura;

III. Análise exploratória: distribuição, correlação, histogramas individuais das variáveis;

IV. Modelagem preditiva: Modelos de *ML* são treinados e avaliados para prever o desempenho estrutural, com foco na otimização de custos e eficiência. Os dados foram divididos em duas partes, sendo 80% utilizados para treinamento do modelo e 20% para teste, garantindo que a avaliação do desempenho fosse realizada em dados não vistos durante o treinamento. Além disso, foi definido um valor fixo para a semente aleatória, permitindo que a divisão dos dados seja reproduzível em futuras execuções do experimento.

Essa abordagem metodológica não apenas amplia as possibilidades de exploração no projeto de estruturas metálicas, mas também estabelece uma base teórica e prática para a aplicação de técnicas computacionais avançadas na Engenharia Civil. Ao combinar a geração de dados com Algoritmos Genéticos e a análise preditiva com *Machine Learning*, este trabalho busca contribuir para o desenvolvimento de soluções estruturais mais inovadoras e sustentáveis.

1.1 Objetivo geral

Desenvolver e validar uma metodologia baseada em Algoritmos Genéticos e técnicas de *Machine Learning* para otimizar o projeto de estruturas metálicas, com foco na redução de custos, melhoria do desempenho estrutural e aumento da sustentabilidade dos projetos.

1.2 Objetivos específicos

I. Automatizar a geração e coleta de dados relevantes para o projeto de estruturas metálicas, integrando ferramentas de modelagem paramétrica, como *Rhinoceros* e *Grasshopper*, com Algoritmos Genéticos;
II. Implementar modelos de *Machine Learning* supervisionados, como *Random Forest*, *Gradient Boosting* e *XGBoost*, para prever o desempenho estrutural e auxiliar na tomada de decisões;
III. Avaliar o desempenho dos modelos preditivos utilizando métricas robustas, como *MAE*, *MSE*, *RMSE* e R^2;
IV. Identificar padrões que possam contribuir para o desenvolvimento de projetos estruturais mais econômicos, eficientes e sustentáveis.

1.3 Justificativa

Desenvolver e validar uma metodologia baseada em Algoritmos Genéticos e técnicas de *Machine Learning* para otimizar o projeto de estruturas metálicas, com foco na redução de custos, melhoria do desempenho estrutural e aumento da sustentabilidade dos projetos.

2 MATERIAL E MÉTODOS

2.1 Geração de dados com Algoritmo Genéticos

O Algoritmo Genético (*GA*, do inglês *Genetic Algorithm*) é um método de otimização inspirado nos princípios da evolução biológica (HOLLAND, 1975). Esse método busca encontrar soluções ótimas ou próximas do ideal para problemas complexos, simulando processos como seleção natural, reprodução e mutação. A seguir, descrevemos as etapas fundamentais do Algoritmo Genético:

I. Geração inicial: o algoritmo inicia com a criação aleatória de uma população inicial de soluções, chamadas de indivíduos. Cada indivíduo é representado como um cromossomo, que codifica uma possível solução para o problema. Essa representação

pode ser binária, numérica ou simbólica, dependendo da natureza do problema (GOLDBERG, 1989);

II. Avaliação: cada indivíduo da população é avaliado por meio de uma função de aptidão (*fitness function*), que quantifica quão bem a solução proposta atende aos objetivos do problema. A função de aptidão é essencial para guiar o processo de seleção e evolução da população (MITCHELL, 1998);

III. Seleção: indivíduos com maior aptidão têm maior probabilidade de serem selecionados para a reprodução. Técnicas comuns de seleção incluem a roleta viciada (*roulette wheel selection*) e a seleção por torneio (*tournament selection*). Esses métodos garantem que soluções mais promissoras tenham maior chance de contribuir para a próxima geração (DEB, 2001);

IV. Cruzamento: os indivíduos selecionados são combinados por meio de operadores de cruzamento (*crossover*), que geram novos indivíduos (filhos) a partir da recombinação do material genético dos pais. O cruzamento simula a reprodução sexual e promove a exploração do espaço de soluções (HOLLAND, 1975);

V. Mutação: para introduzir diversidade genética e evitar a convergência prematura para ótimos locais, operadores de mutação são aplicados aleatoriamente a alguns indivíduos. A mutação altera pequenas partes do cromossomo, permitindo a exploração de novas regiões do espaço de soluções (GOLDBERG, 1989);

VI. Repetição: os passos de avaliação, seleção, cruzamento e mutação são repetidos por várias gerações até que um critério de parada seja atingido. Esse critério pode ser um número máximo de gerações, a estagnação da aptidão média da população ou a obtenção de uma solução satisfatória (DEB, 2001).

Em resumo, o Algoritmo Genético é uma estratégia eficaz para resolver problemas de otimização complexos, especialmente em cenários onde o espaço de busca é vasto e não-linear. Sua capacidade de explorar múltiplas soluções simultaneamente e de escapar de ótimos locais o torna amplamente aplicável em áreas como Engenharia, Ciência da Computação e Pesquisa Operacional (MITCHELL, 1998).

No contexto deste trabalho, cada configuração estrutural foi tratada como um indivíduo dentro de uma população, sendo avaliada quanto ao desempenho estrutural com base em métricas como peso, estabilidade e custo. Esse processo iterativo gerou um grande volume

de dados variados, que foram posteriormente utilizados para a aplicação de técnicas de *Machine Learning*, visando a otimização de estruturas metálicas.

2.2 Modelagem Paramétrica Estrutural

A modelagem paramétrica estrutural permite criar estruturas flexíveis e complexas a partir de princípios simples (BARBOSA, 2018), transformando um design estático em um sistema dinâmico. Seguindo essa abordagem, foi desenvolvido um modelo de cobertura treliçada, estruturado a partir de um conjunto de parábolas interligadas por diagonais.

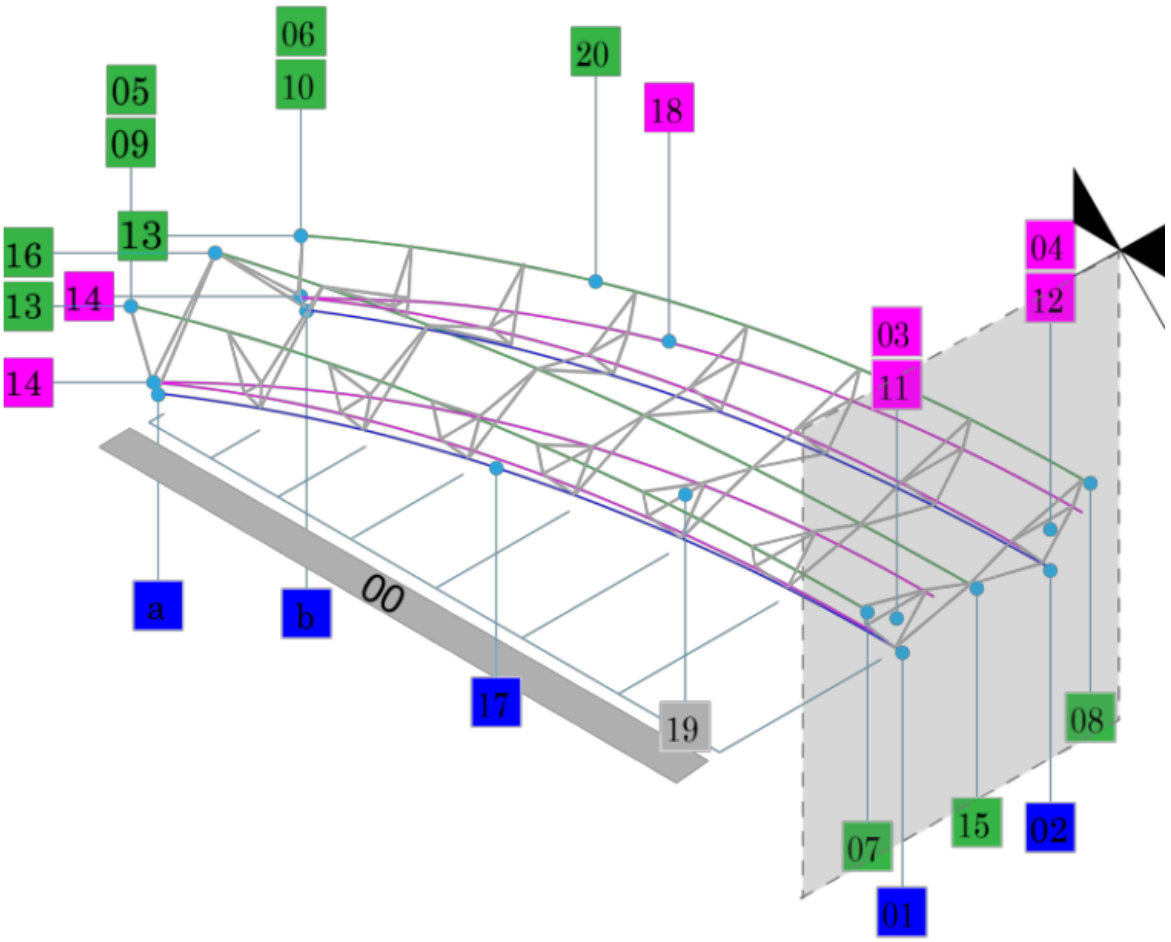
A estrutura é composta por quatro grupos:

I. Barras A (sinalizadas em azul, na Figura 1): formadas pelas parábolas da base e controladas por parâmetros no eixo z do ponto médio, com extremidades apoiadas nos pontos *a* e *b*;
II. Barras B (sinalizadas em rosa, na Figura 1): controladas nos eixos *z* e *y* no ponto médio e nos eixos *x* e *z* nas extremidades;
III. Barras C (sinalizadas em verde, na Figura 1): ajustadas pelos mesmos eixos que as barras B, mas com configurações distintas;
IV. Barras D (sinalizadas em cinza, na Figura 1): formadas pelas diagonais, elas conectam os pontos de interseção das parábolas.

Cada conjunto de barras possui um parâmetro específico que define seu perfil estrutural, baseado em uma tabela de seções tubulares circulares da empresa *Vallourec*, fabricante nacional de perfis metálicos.

A modelagem paramétrica estabelece relações entre os 21 parâmetros do projeto, com valores variando de 10 a 500 caracteres, permitindo gerar diferentes configurações formais, o que garante ampla diversidade de soluções.

Figura 1 - Diagrama do projeto e parâmetros de entrada



Fonte: Elaboração própria, 2024.

2.3 Coleta dos dados

Os dados foram coletados a partir dos resultados gerados pelo modelo paramétrico na aplicação do *GA*, processo que simulou múltiplas soluções para o projeto da estrutura metálica. A cada iteração, os operadores genéticos (seleção, crossover e mutação) foram aplicados para evoluir as soluções, registrando tanto os valores de entrada quanto os resultados obtidos.

Os valores de entrada da modelagem paramétrica e seus respectivos valores de saída foram armazenados em arquivos .CSV (*Comma-Separated Values*), permitindo a análise posterior das relações entre os *inputs* e as características finais da estrutura.

2.4 Análises exploratória dos dados

A análise exploratória será realizada sobre o conjunto de dados gerado pelo Algoritmo Genético e capturado pelo algoritmo de coleta de dados. O *dataset* final contém 5.504 registros e 22 variáveis, resultantes da simulação de 100 gerações com 59 indivíduos por geração.

A estrutura do conjunto de dados inclui variáveis de entrada, relacionadas às características estruturais e geométricas, e variáveis de saída, associadas ao desempenho da estrutura. Para explorar a distribuição e correlação das variáveis, serão utilizados histogramas individuais para as 22 variáveis explicativas e um mapa de calor (*heatmap*) de correlações.

2.5 Modelagem via *Machine Learning*

Modelos supervisionados (*Random Forest*, *Gradient Boosting* e *XGBoost*) foram utilizados para prever o peso das estruturas com base nos dados gerados. Para garantir uma avaliação robusta, o conjunto de dados foi dividido em 80% para treinamento e 20% para teste, utilizando um valor fixo de *random state* igual a 42, assegurando a reprodutibilidade dos experimentos. A variável dependente foi definida como goal, enquanto as variáveis independentes consistiram nas demais colunas do conjunto de dados.

Para melhorar o desempenho dos modelos e garantir que os atributos estivessem na mesma escala, os dados foram padronizados por meio da transformação *Standard Scaler*, aplicada separadamente aos conjuntos de treinamento e teste. Nenhum ajuste de hiperparâmetros foi realizado, mantendo os modelos em sua configuração padrão.

A avaliação do desempenho preditivo foi conduzida utilizando métricas amplamente aceitas na literatura:

I. Erro absoluto médio (*MAE*);
II. Erro Quadrático Médio (*MSE*);
III. Raiz do erro quadrático médio (*RMSE*);
IV. Coeficiente de determinação (R^2).

Essa abordagem integrou coleta de dados, análises estatísticas e *Machine Learning* de forma coesa, promovendo rigor metodológico e confiabilidade nos resultados.

3 RESULTADOS E DISCUSSÃO

3.1 Análise Exploratória dos Dados

A geração da massa de dados foi realizada por meio da aplicação de um Algoritmo Genético, conforme descrito anteriormente. O processo seguiu a seguinte configuração:

I. número de gerações igual a 100;
II. população de 59 indivíduos por geração;
III. taxa de mutação de 0.05;
IV. objetivo principal de minimizar o peso total da estrutura metálica.

Com base nesses parâmetros, o tamanho esperado da amostra seria de 5.900 observações (59 indivíduos por 100 gerações). No entanto, devido a falhas operacionais desconhecidas do algoritmo de coleta de dados, foi possível capturar apenas 5.504 linhas, resultando em um dataset final composto por 5.504 registros e 22 variáveis.

A estrutura do conjunto de dados reflete a complexidade do problema de otimização, contendo variáveis de entrada relacionadas às características estruturais e geométricas e variáveis de saída associadas ao desempenho da estrutura, representada pelo seu peso.

A Figura 2 apresenta uma matriz de correlação de Pearson, onde cada célula representa o coeficiente de correlação entre duas variáveis. A escala de cores, variando do azul (para correlações negativas) até o vermelho (para correlações positivas), permite visualizar a intensidade e direção das relações.

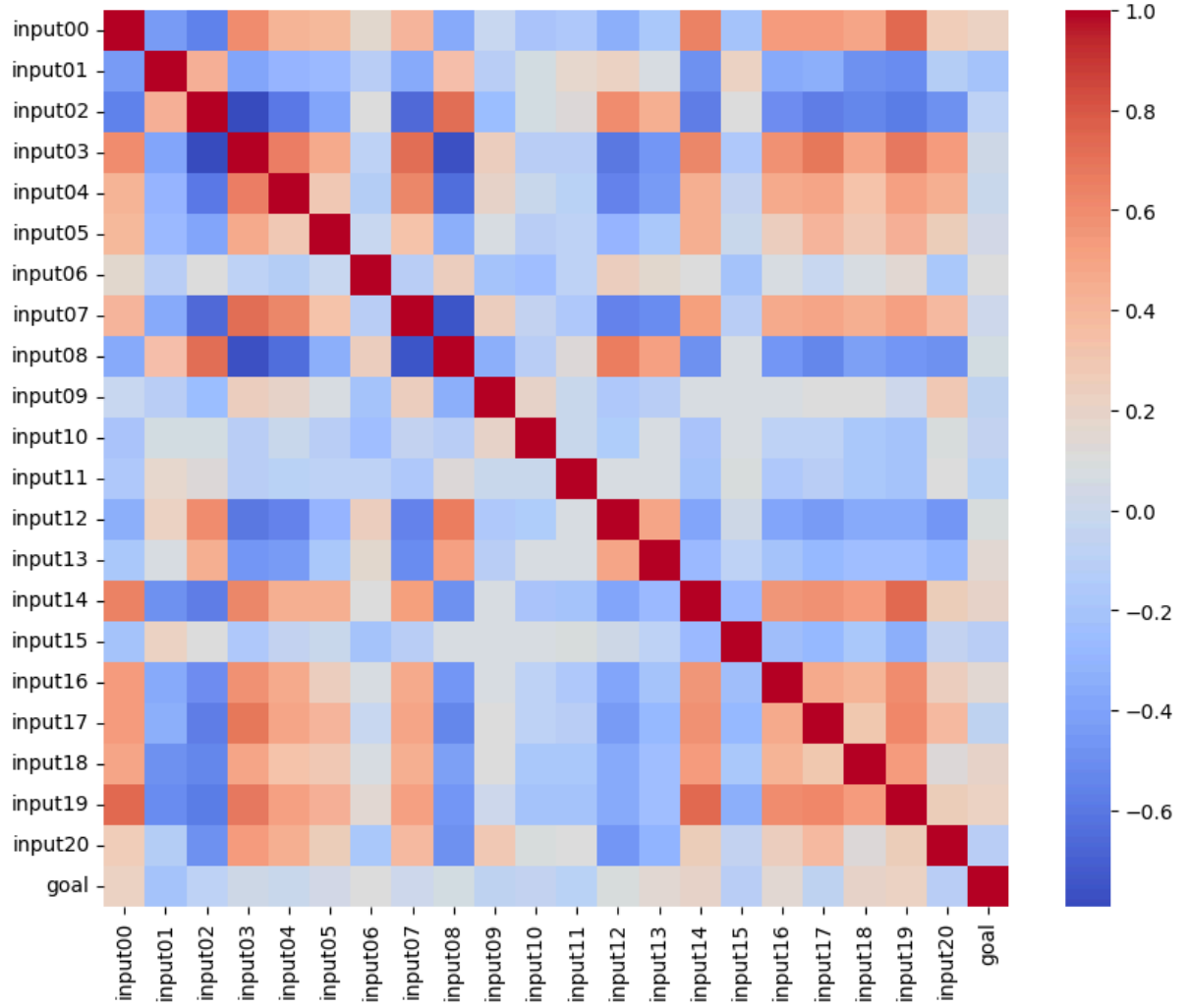
Observa-se uma diagonal principal de cor vermelha intensa, indicando uma autocorrelação perfeita (coeficiente de correlação igual a 1) de cada variável consigo mesma. Adicionalmente, algumas variáveis apresentam correlações positivas elevadas entre si, como por exemplo, o grupo de variáveis input00 a input04.

A matriz também revela algumas correlações negativas, representadas por tons de azul. Estas indicam que o aumento em uma variável está associado à diminuição na outra. Um exemplo é a relação entre input14 e goal, que nesse caso é a nossa variável resposta (em um contexto de *Machine Learning*, é comum se referir à variável resposta/dependente como o "objetivo"), que exibe uma correlação negativa moderada.

A análise visual da matriz revela a existência de padrões de correlação entre grupos de variáveis. Estes padrões podem indicar relações subjacentes ou fatores comuns influenciando o comportamento das variáveis.

A última coluna da matriz mostra as correlações de cada variável com a variável *goal*. Esta informação é particularmente relevante, pois permite identificar quais variáveis estão mais fortemente relacionadas à variável resposta/objetivo em questão.

Figura 2 - Gráfico *heatmap* de correlação entre as variáveis

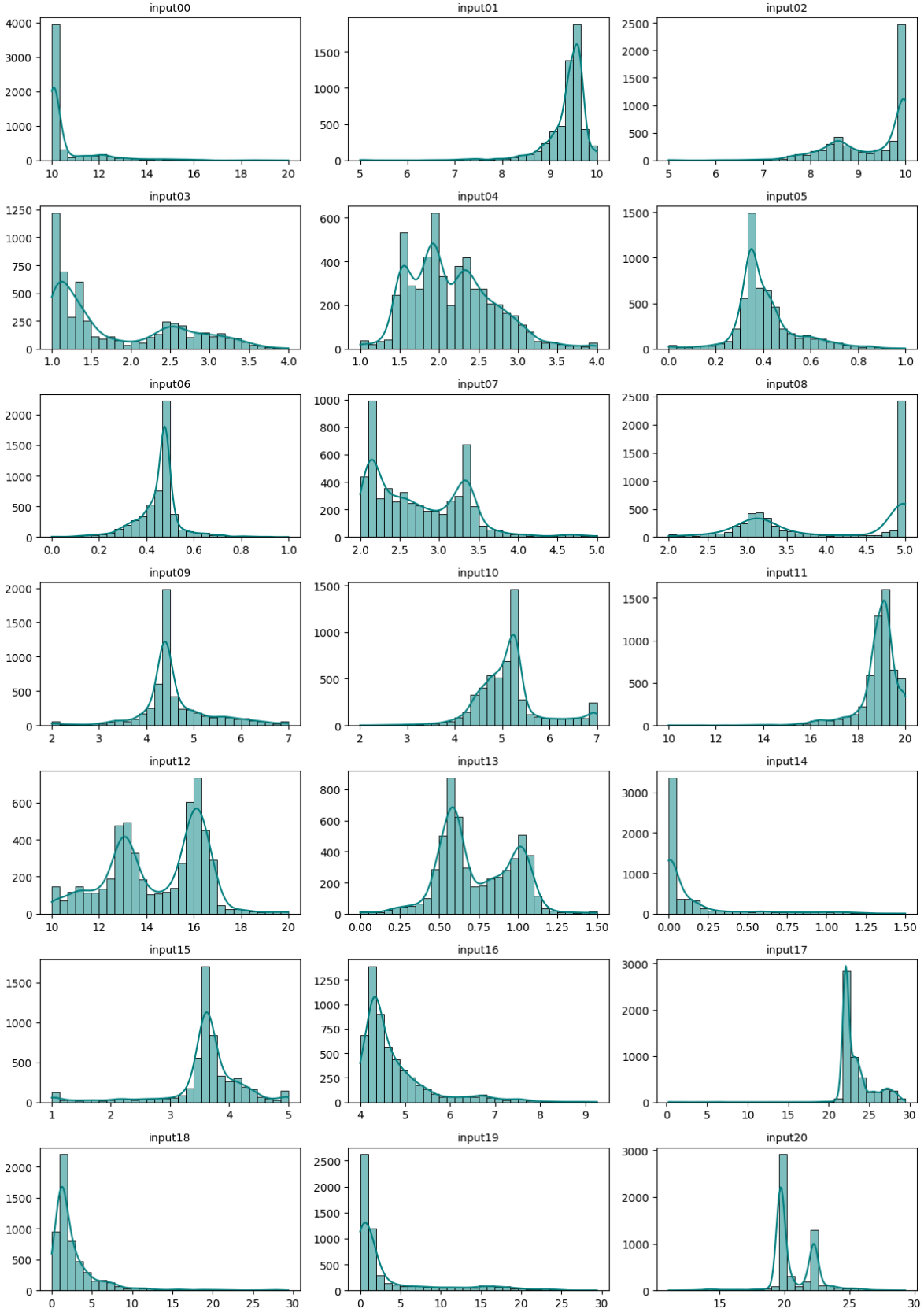


Fonte: Elaboração própria, 2024.

A Figura 3 apresenta um conjunto de 21 histogramas, cada um representando a distribuição de frequência de uma variável individual. Adicionalmente, um gráfico de densidade (linha contínua) foi sobreposto a cada histograma, fornecendo uma estimativa da forma da

distribuição subjacente. A análise visual dos histogramas revela uma variedade de formas de distribuição. Algumas variáveis apresentam distribuições aproximadamente normais (em forma de sino), como input02 e input17.

Figura 3 - Histogramas para as 21 variáveis explicativas



Fonte: Elaboração própria, 2024.

Por outro lado, algumas variáveis exibem assimetria positiva (cauda longa à direita), como input03 e input14, ou até assimetria negativa (cauda longa à esquerda), como input12. Há também casos de distribuições bimodais (dois picos), como input07 e input13, sugerindo a presença de duas populações distintas ou de efeitos de mistura.

Em alguns histogramas, como input14 e input18, observa-se a presença de outliers, ou seja, valores muito distantes da maioria dos dados. Estes outliers podem ser erros de medição, eventos raros ou características genuínas da distribuição.

3.2 Análise dos Resultados dos Modelos de *Machine Learning*

Para prever o peso das estruturas metálicas, foram utilizados três modelos de *Machine Learning*. O *Gradient Boosting*, modelo de aprendizado supervisionado baseado em árvores de decisão sequenciais, onde cada nova árvore corrige os erros da anterior. Esse modelo é eficaz para capturar relações complexas nos dados e minimizar erros residuais. O *XGBoost*, variante otimizada do *Gradient Boosting* que utiliza regularização para evitar overfitting e técnicas de paralelização para acelerar o treinamento. Este modelo é amplamente utilizado em competições de *Machine Learning* devido ao seu alto desempenho. Por fim, o *Random Forest*, algoritmo de aprendizado baseado em um conjunto de múltiplas árvores de decisão treinadas independentemente. A previsão final deste modelo é obtida por meio da média ou da moda das previsões individuais, tornando-o robusto contra o *overfitting*.

Avaliamos o desempenho desses modelos utilizando as seguintes métricas:

I. Erro Absoluto Médio (*MAE*), que quantifica a média dos erros absolutos entre as previsões ($\widehat{yi}$) e os valores reais (yi), proporcionando uma interpretação intuitiva da magnitude dos erros. Sua fórmula é dada pela equação 1:

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

II. Raiz do Erro Quadrático Médio (*RMSE*), que corresponde à raiz quadrada do MSE, permitindo melhor interpretação ao manter a unidade original dos dados. Sua fórmula é dada pela equação 2:

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

III. Erro Quadrático Médio (*MSE*), que representa a média dos quadrados dos erros, atribuindo maior penalização a erros elevados, conforme a equação 3:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (3)$$

IV. Coeficiente de Determinação (R^2), que expressa a proporção da variabilidade dos dados explicada pelo modelo, sendo definido pela equação 4:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (4)$$

Onde, $\widehat{y}$ representa a média dos valores dos yi.

Essas métricas foram aplicadas para comparar o desempenho dos modelos, garantindo que a escolha final seja baseada em critérios quantitativos objetivos.

3.3 *Random Forest*

O modelo *Random Forest Regressor* obteve um R^2 de 0.2535, o que indica que ele explica apenas 25% da variabilidade dos dados, um desempenho inferior ao esperado para um modelo de *ensemble*.

Os erros foram:

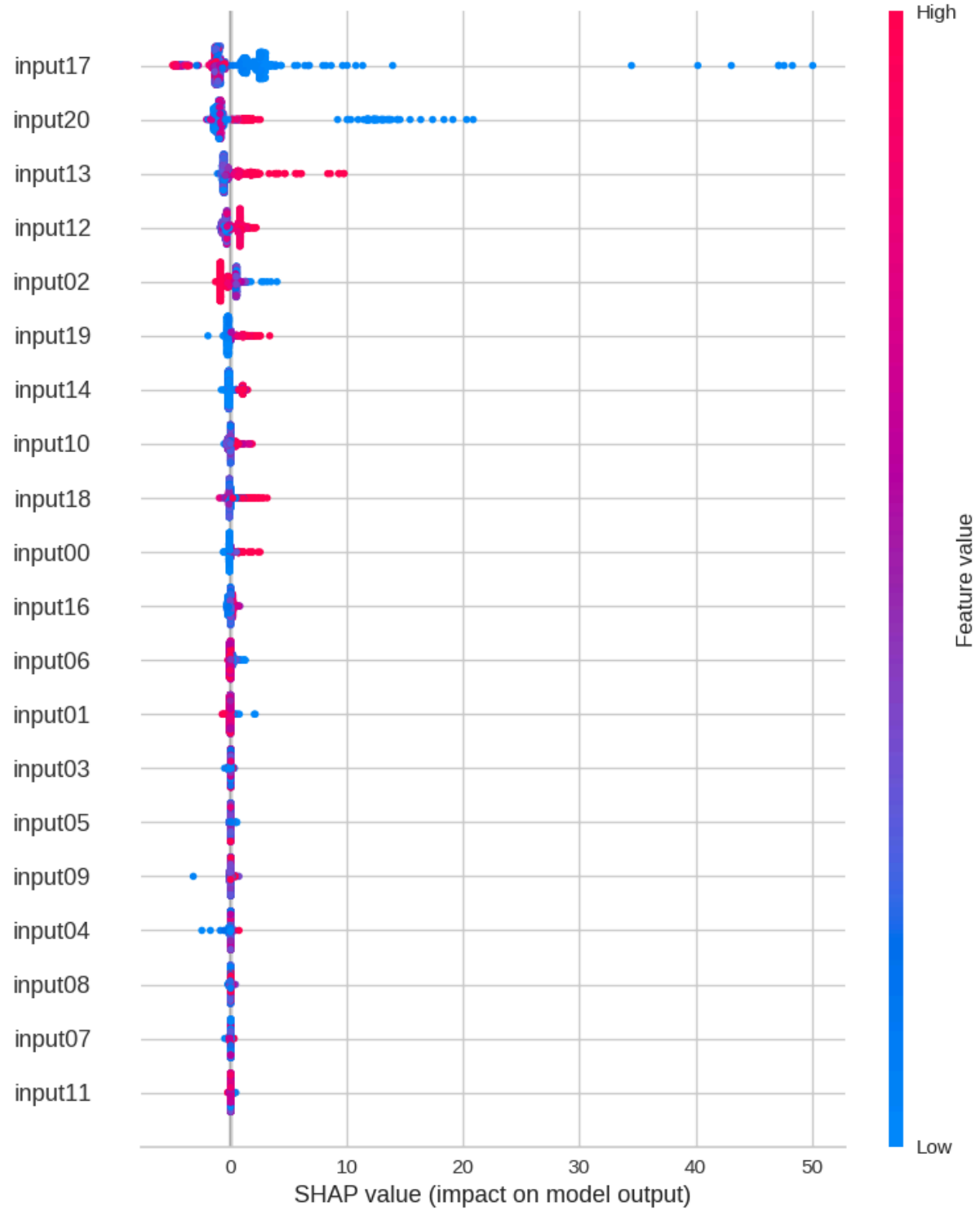
I. *MAE* de 7.0408;
II. *MSE* de 96.3182;
III. *RMSE* de 9.8142;
IV. O baixo R^2 e os erros elevados sugerem que o modelo pode estar subajustado (*underfitting*), possivelmente devido à falta de profundidade nas árvores ou à necessidade de um melhor ajuste de hiperparâmetros.

Para entendermos melhor as predições individuais, elaboramos um diagrama *LIME* (*Local Interpretable Model-Agnostic Explanations*). Neste gráfico, é apresentada a decomposição da previsão do modelo *Random Forest*, evidenciando quais variáveis tiveram influência positiva ou negativa na predição. O *LIME* permite entender como um modelo de *Machine Learning* toma decisões localmente, e contém detalhes da decomposição da previsão de um modelo, evidenciando quais variáveis tiveram influência positiva ou negativa na predição.

Na Figura 4, podemos notar que as variáveis input17 e input20 exibem maior dispersão dos valores *SHAP*, indicando que, assim como no modelo anterior, possuem um impacto maior na predição. Em contraste, variáveis como input07 e input11 apresentam distribuição mais concentrada ao redor de zero, sugerindo menor influência no modelo.

Sobre a direção do efeito das features na predição, notamos que para input17, valores elevados (em vermelho) deslocam-se, mais uma vez, para a direita no eixo X (sugerindo que esses então aumentam a predição) enquanto valores baixos (em azul) possuem uma tendência oposta.

Figura 4 - Diagrama de impacto de cada variável na previsão do modelo *Random Forest*

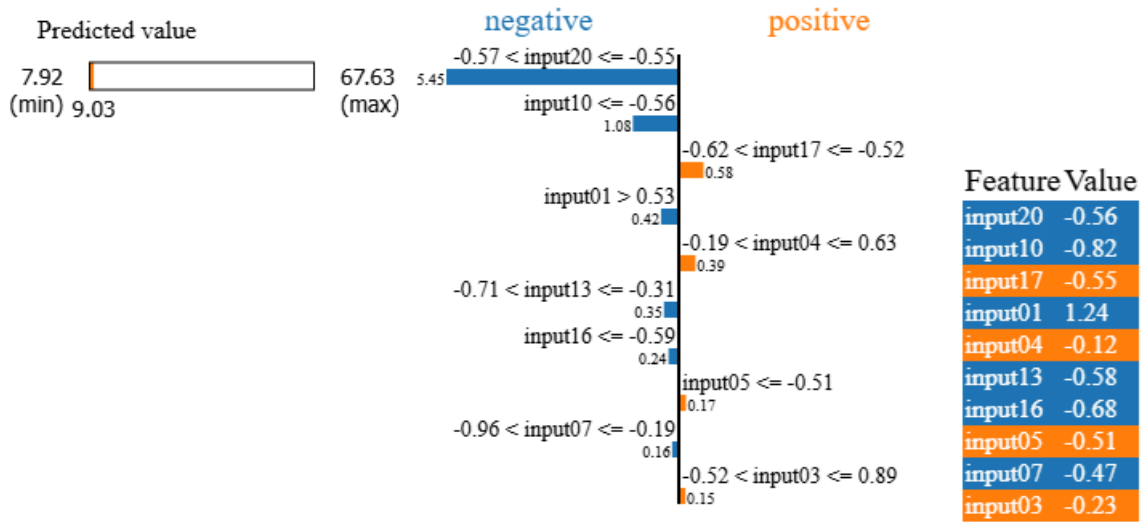


Fonte: Elaboração própria, 2024.

Ainda na Figura 4, algumas variáveis apresentam padrões simétricos, enquanto outras, como input20, demonstram comportamento assimétrico, sugerindo uma relação não-linear entre seus valores e a predição do modelo.

Na Figura 5, notamos que o valor predito do modelo *Random Forest* foi de 9.03, dentro do intervalo esperado, cujo mínimo é 7.92 e o máximo é 67.63.

Figura 5 - Diagrama LIME do modelo *Random Forest*



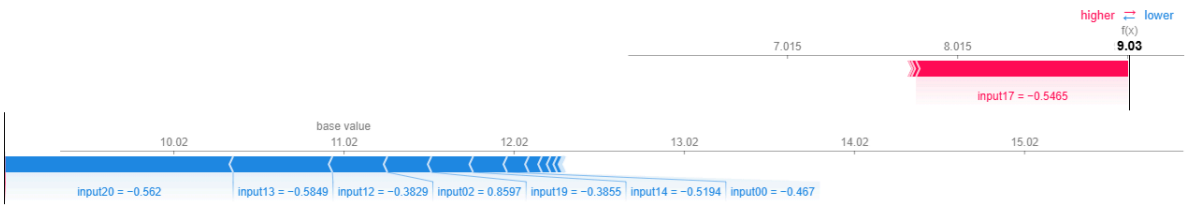
Fonte: Elaboração própria, 2024.

Ainda na Figura 5, podemos notar que as variáveis que reduziram a predição são input20, input10, input01, input13, input16 e input07, sendo input20 a mais influente nessa direção. Por outro lado, as variáveis input17, input04, input05 e input03 tiveram os maiores impactos positivos, contribuindo para um aumento da previsão.

Sobre os valores das *features*, notamos que input17 possui valor -0.56, que input10 tem -0.82, e que input01 apresenta 1.24, o que significa que diferentes magnitudes influenciaram a decisão deste modelo.

Finalmente, na Figura 6 notamos que este modelo tem um valor de referência (base value) = 11.02 mas o modelo previu 9,03, o que significa que as contribuições das features aumentaram o valor a partir do valor base.

Figura 6 - Diagrama explicação global da predição do modelo *Random Forest*.



Fonte: Elaboração própria, 2024.

As variáveis que reduziram a predição foram:

I. input17 = -0.5465.

Por outro lado, as variáveis que aumentaram a predição foram:

I. input20 = -0.562;
II. input13 = -0.5829;
III. input12 = -0.3829;
IV. input02 = 0.8597;
V. input19 = -0.3855;
VI. input14 = -0.5194;
VII. input00 = -0.467.

Essas variáveis tiveram um impacto positivo, mas não suficiente para ultrapassar as influências negativas.

Na interpretação geral, a feature input17 teve um impacto significativo para diminuir a previsão abaixo do valor base. Já as features input20, input13, input12, input19, input14 e input00, tentaram aumentar o valor, mas o impacto coletivo das variáveis vermelhas foi maior, resultando na previsão final de 9.03.

3.4 *Gradient Boosting*

O modelo *Gradient Boosting* apresentou um R^2 de 0.5844, indicando que ele explica cerca de 58% da variabilidade dos dados, um desempenho sólido para o problema em questão.

Os erros foram:

I. *MAE* de 4.7674;

II. *MSE* de 53.6243;

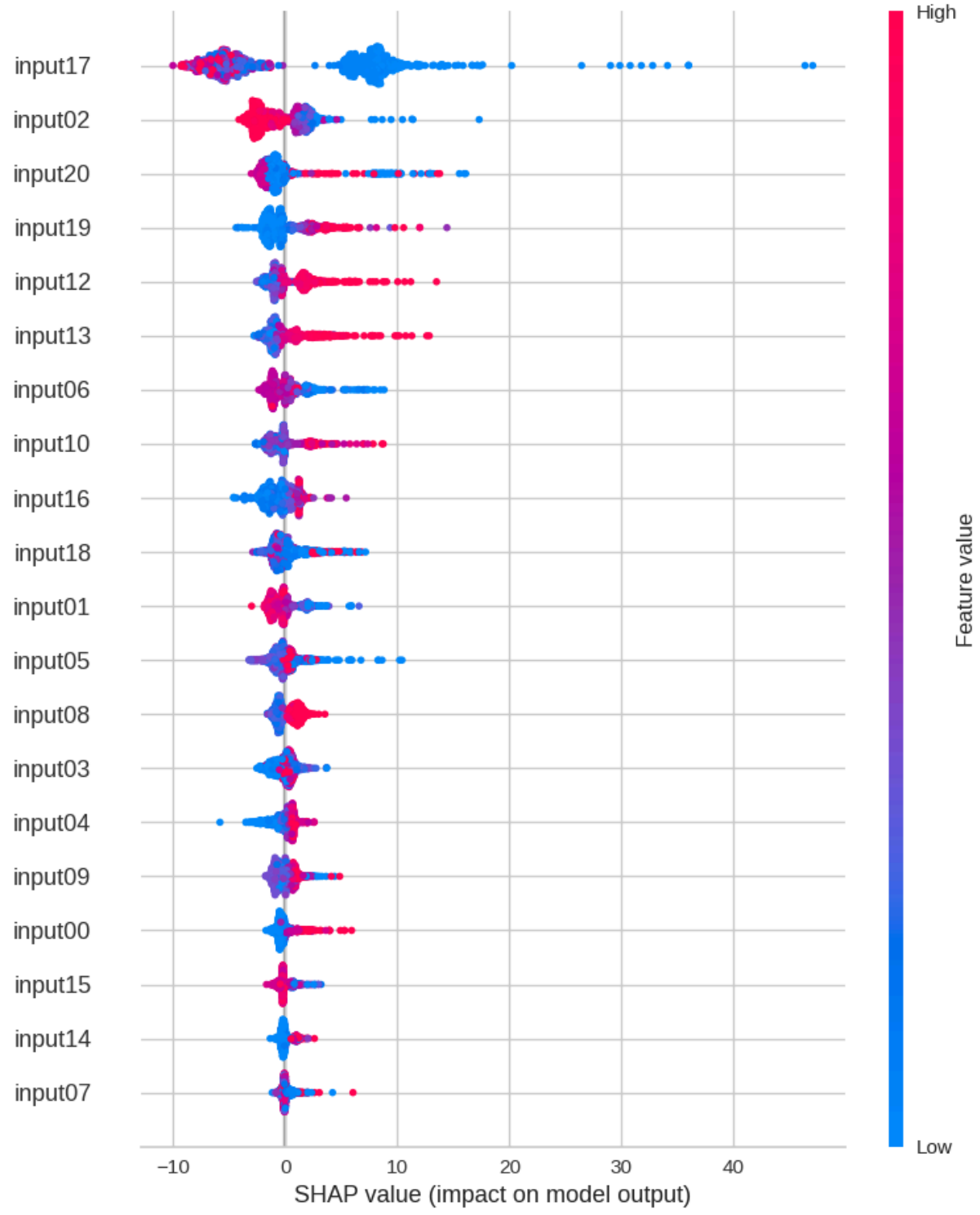
III. *RMSE* de 7.3229.

Os resultados indicam que o *Gradient Boosting* foi eficaz em capturar os padrões dos dados, alcançando um bom equilíbrio entre erro e capacidade preditiva.

Para uma análise mais aprofundada do comportamento do modelo e da influência de cada variável na predição, utilizamos a explicabilidade via *SHAP* (do inglês, *Shapley Additive Explanations*). Essa abordagem permite compreender o impacto individual de cada atributo/*feature* na saída do modelo, fornecendo maior conhecimento sobre sua importância e o comportamento das relações não lineares.

A Figura 7 apresenta o *SHAP* para um modelo *Gradient Boosting*, e nos permite identificar padrões e relações entre as variáveis de entrada e saída do modelo.

Figura 7 - Diagrama de impacto de cada variável na previsão do modelo *Gradient Boosting*



Fonte: Elaboração própria, 2024.

As variáveis input17 e input02 exibem maior dispersão dos valores *SHAP*, indicando que possuem impacto significativo na predição. Em contraste, variáveis como input14 e input07

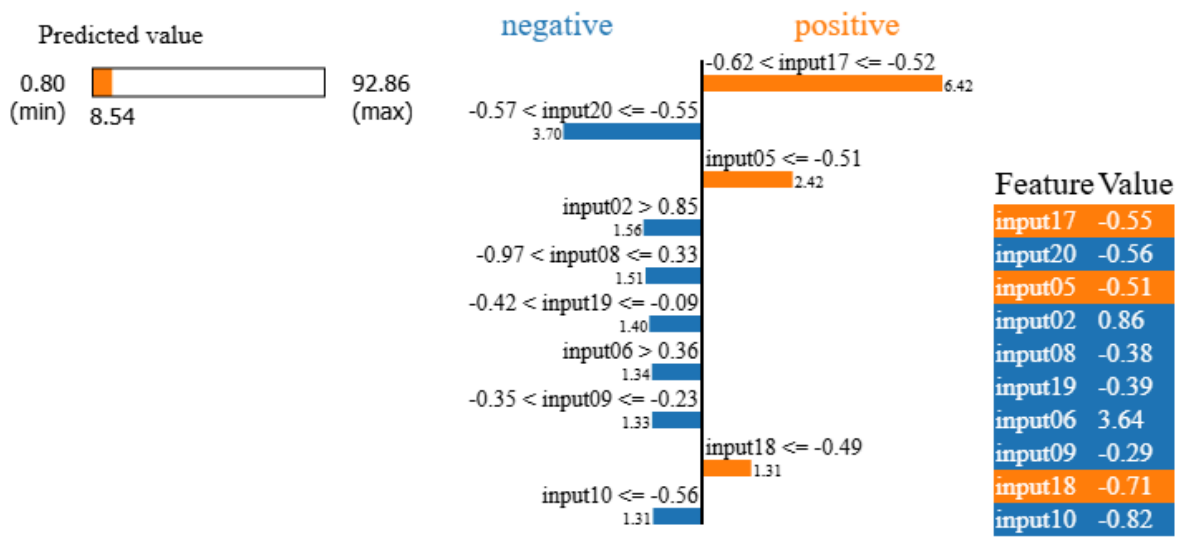
apresentam distribuição mais concentrada ao redor de zero, sugerindo menor influência no modelo.

No *SHAP*, a relação entre os valores da variável e seu impacto pode ser observada por meio das cores. Por exemplo, para input17, valores elevados (em vermelho) deslocam-se para a direita no eixo x, sugerindo que aumentam o valor da predição, enquanto valores baixos (em azul) possuem uma tendência oposta.

Ainda na Figura 7, algumas features mostram distribuição simétrica, enquanto outras têm uma assimetria forte (ex: input17 e input02), sugerindo comportamento não linear no modelo.

A Figura 8 contém o diagrama *LIME* para o modelo *Gradient Boosting*, evidenciando quais variáveis tiveram influência positiva ou negativa na predição. Podemos notar que o valor predito do modelo é 8.54, dentro do intervalo cujo mínimo é 0.80 e o máximo é 92.86.

Figura 8 - Diagrama LIME do modelo *Gradient Boosting*



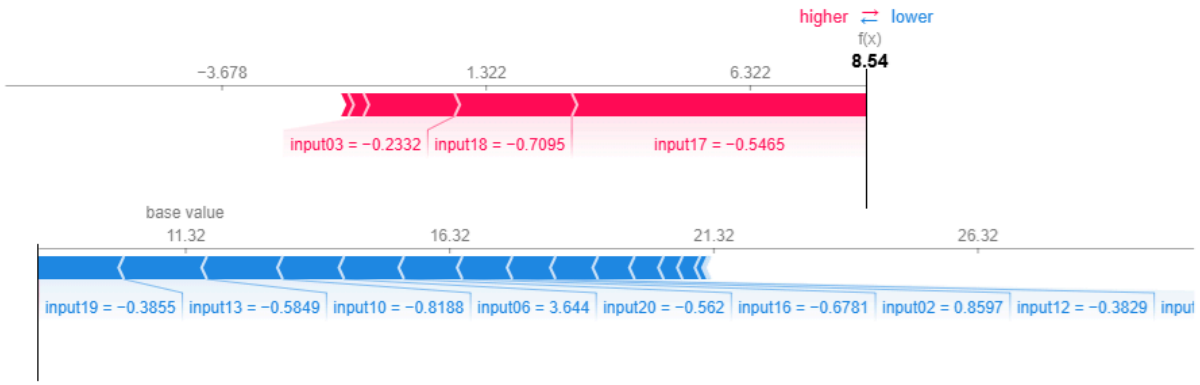
Fonte: Elaboração própria, 2024.

Ainda na Figura 8, conseguimos analisar as contribuições negativas e positivas de cada feature. As barras em azul indicam variáveis que reduziram a previsão, enquanto as barras em laranja indicam variáveis que aumentaram a previsão. Dessa forma, podemos notar que as variáveis que reduziram a predição são input20, input02, input08, input19, input06, input09, input10, sendo input20 a mais influente nessa direção. Já as variáveis que

aumentaram a predição são input17, input05 e input18, sendo input17 a que mais contribuiu para um aumento da previsão. Sobre os valores das features, notamos que input17 possui valor -0.55, que input05 tem -0.51, e que input02 apresenta 0.86, o que significa que diferentes magnitudes influenciaram a decisão do modelo.

Na Figura 9 temos o gráfico de *SHAP* values que fornece uma explicação da predição do modelo de forma mais global e consistente em relação ao impacto das variáveis. O modelo tem um valor de referência (base value) de 11.32; esse é o valor médio da predição quando nenhuma *feature* específica é considerada. O modelo previu 8.54, o que significa que as contribuições das features proporcionaram uma redução do valor do *goal* relativo ao valor base (o que é desejável em termos do nosso problema, uma vez que nosso objetivo original é essencialmente otimizar as estruturas reduzindo seu peso).

Figura 9 - Gráfico de SHAP values do modelo *Gradient Boosting*



Fonte: Elaboração própria, 2024.

As variáveis que reduziram a predição (cor vermelha – impacto positivo na diminuição do valor) são:

I. input03 = -0.2332;
II. input18 = -0.7095;
III. input17 = -0.5465.

Por outro lado, as variáveis que aumentaram a predição (cor azul – impacto positivo no aumento do valor) foram:

I. input19 = -0.3855;

II. input13 = -0.5849;
III. input10 = -0.8188;
IV. input06 = 3.644;
V. input20 = -0.562;
VI. input16 = -0.6781;
VII. input02 = 0.8597;
VIII. input12 = -0.3829;
IX. input09 = -0.2863;
X. input15 = 0.1096

Essas variáveis tiveram um impacto positivo no valor predito, mas sua magnitude não foi suficiente para ultrapassar as influências negativas.

Na interpretação geral, as *features* input03, input18 e input17 tiveram um impacto importante para diminuir a previsão abaixo do valor base. Já as features input06, input02 e input15 tentaram aumentar o valor, mas o impacto coletivo das variáveis vermelhas foi maior, resultando na previsão final de 8.54.

3.5 *XGBoost*

O modelo *XGBoost* treinado apresentou um R^2 de 0.5809, indicando que ele explica cerca de 58% da variabilidade dos dados.

Os erros foram:

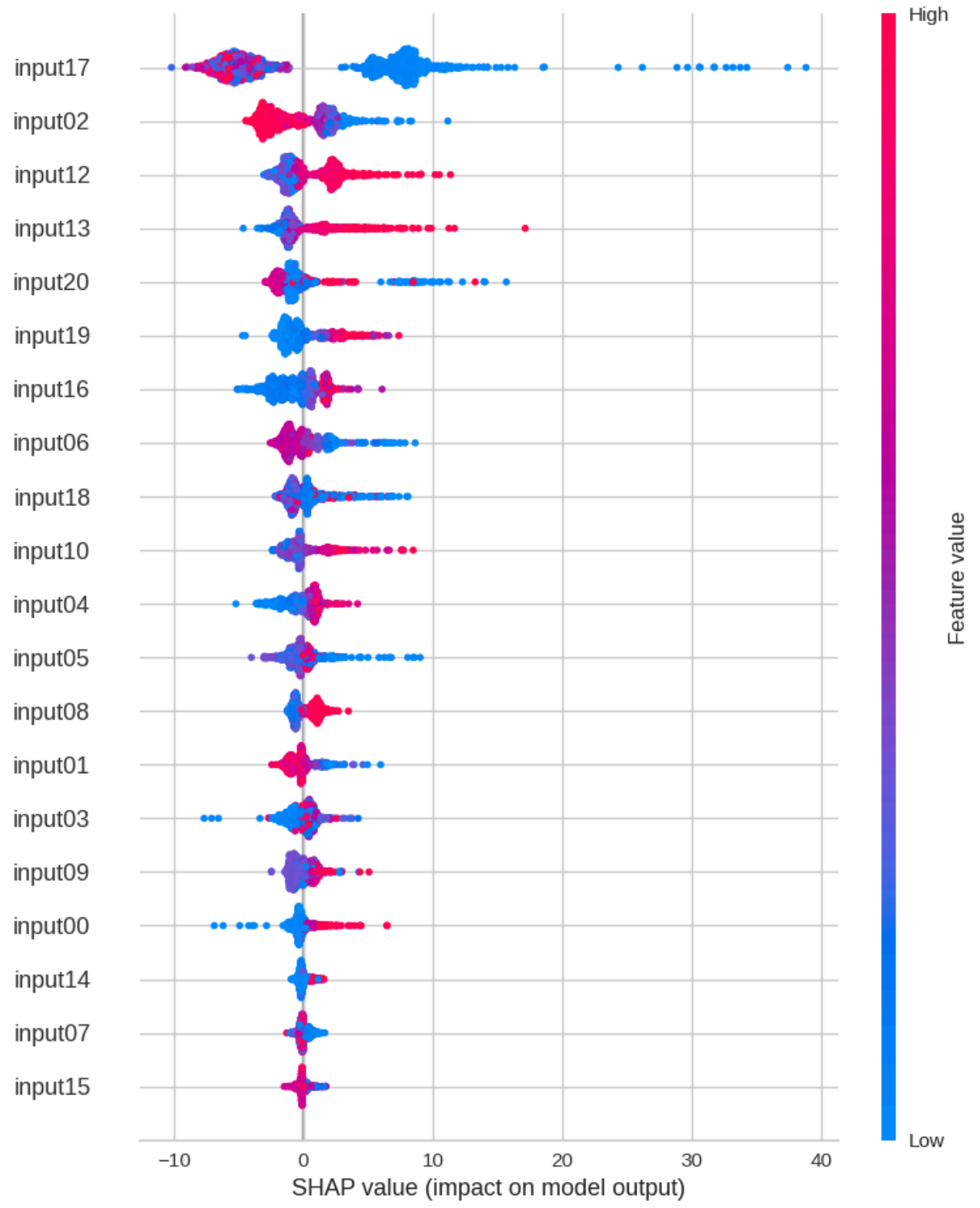
IV. *MAE* igual a 4.7255;
V. *MSE* de 54.0651;
VI. *RMSE* de 7.3529.

Embora o modelo tenha um desempenho razoável, a margem de erro sugere que há espaço para otimizações, como ajuste de hiperparâmetros, *feature engineering* ou técnicas para reduzir viés e variância.

Na Figura 10, podemos notar que as variáveis input17 e input02 exibem maior dispersão dos valores *SHAP*, indicando que, assim como no modelo anterior, possuem impacto significativo na predição. Em contraste, variáveis como input07 e input15 apresentam distribuição mais concentrada ao redor de zero, sugerindo menor influência no modelo.

Sobre o efeito da magnitude das *features*, notamos que para input17, valores elevados (vermelho) deslocam-se para a direita no eixo X, sugerindo que aumentam a predição, enquanto valores baixos (azul) possuem tendência oposta.

Figura 10 - Diagrama de impacto de cada variável na previsão do modelo *XGBoost*

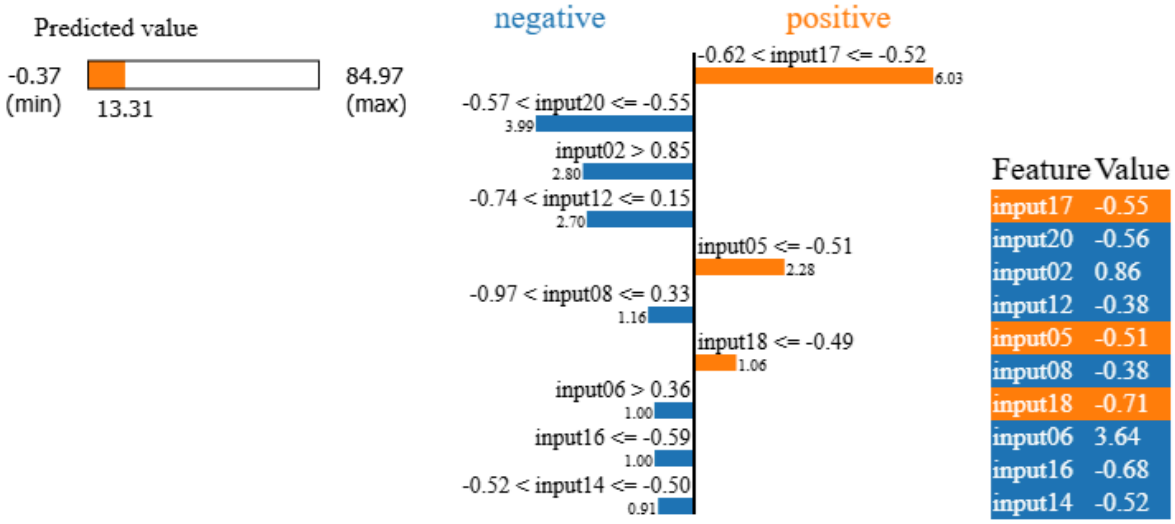


Fonte: Elaboração própria, 2024.

Algumas variáveis apresentam padrões simétricos, enquanto outras, como input02, demonstram comportamento assimétrico, sugerindo uma relação não-linear entre seus valores e a predição do modelo.

Na Figura 11, notamos que o valor predito do modelo *XGBoost* foi de 13.31, dentro do intervalo esperado, cujo mínimo é -0.37 e o máximo é 84.97.

Figura 11 - Diagrama *LIME* do modelo XGBoost

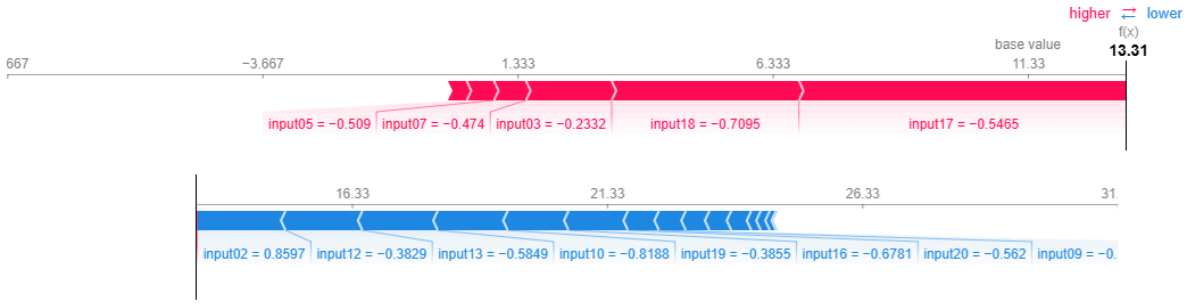


Fonte: Elaboração própria, 2024.

Ainda na Figura 11, podemos notar que as variáveis que reduziram a predição são input20, input02, input12, input08, input06, input16, input14, sendo input20 a mais influente nessa direção. Por outro lado, as variáveis input17, input05 e input18 tiveram os maiores impactos positivos, contribuindo para um aumento da previsão. Sobre os valores das features, notamos que input17 possui valor -0.55, que input05 tem -0.51, e que input02 apresenta 0.86, o que significa que diferentes magnitudes influenciaram a decisão deste modelo.

Na Figura 12, notamos que o *XGBoost* tem um valor de referência (base value) igual a 11.33 mas o modelo previu 13.31, o que significa que as contribuições das features aumentaram o valor a partir do valor base.

Figura 12 - Gráfico de *SHAP values* do modelo *XGBoost*



Fonte: Elaboração própria, 2024.

As variáveis que reduziram a predição foram:

I. input05 = -0.509;
II. input12 = -0.474;
III. input03 = -0.2332;
IV. input18 = -0.7095;
V. input17 = -0.5465.

Por outro lado, as variáveis que aumentaram a predição foram:

I. input02 = 0.8597;
II. input12 = -0.3829;
III. input13 = -0.5849;
IV. input10 = -0.8188;
V. input19 = -0.3855;
VI. input16 = -0.6781;
VII. input20 = -0.562;
VIII. input09 = -0.

Essas variáveis tiveram um impacto positivo, mas seu impacto não foi suficiente para ultrapassar as influências negativas.

Na interpretação geral, as *features* input03, input12, input13, input18 e input17 tiveram um impacto relevante para diminuir a previsão abaixo do valor base. Já as *features* input02, input12, input13, input10, input19, input16, input20 e input09, tentaram aumentar o valor, mas o impacto coletivo das variáveis vermelhas foi maior, resultando na previsão final de 13.31.

A Tabela 1 apresenta uma comparação do desempenho dos três algoritmos de *Machine Learning* considerados neste trabalho. Aqui são resumidas as quatro métricas de desempenho reportadas anteriormente, para cada modelo: Erro Médio Absoluto (*MAE*), Erro Quadrático Médio (*MSE*), Raiz do Erro Quadrático Médio (*RMSE*) e Coeficiente de Determinação (*R^2 Score*).

Tabela 1 - Resultados dos ajustes dos modelos de *Machine Learning*

Modelo	***MAE***	***MSE***	***RMSE***	***R^2 Score***
Gradient Boosting	4.7674	53.6243	7.3229	0.5844
XGBoost	4.7255	54.0651	7.3529	0.5809
Random Forest	7.0408	96.3182	9.8142	0.2535

Fonte: Elaboração própria, 2024.

Os resultados indicam que os modelos baseados em boosting, isto é, *Gradient Boosting* e *XGBoost*, apresentaram um desempenho superior em comparação ao *Random Forest* em todas as métricas analisadas. O *Gradient Boosting* obteve um *MAE* de 4.7674, um *MSE* de 53.6243 e um *RMSE* de 7.3229, enquanto o *XGBoost* apresentou valores ligeiramente melhores para *MAE* (4.7255), mas um *MSE* ligeiramente superior (54.0651) e um *RMSE* de 7.3529. Ambos os modelos alcançaram valores próximos do *R^2 Score* (0.5844 para *Gradient Boosting* e 0.5809 para *XGBoost*), indicando que possuem capacidade semelhante de explicação da variância dos dados. Esses valores sugerem que os modelos baseados em boosting conseguem capturar melhor a estrutura dos dados em comparação ao *Random Forest*.

O *Random Forest*, por sua vez, demonstrou um desempenho significativamente inferior, com um *MAE* de 7.0408, *MSE* de 96.3182 e *RMSE* de 9.8142. O valor do *R^2 Score* para este modelo foi de 0.2535, o que indica uma menor capacidade preditiva em relação aos demais algoritmos. Esse resultado pode ser atribuído à menor capacidade do *Random Forest* de modelar relações complexas e interações entre variáveis quando comparado a técnicas de boosting, que são mais eficazes na redução de viés e variância do modelo.

A Tabela 2 apresenta as variáveis que reduziram a predição dos modelos *Random Forest*, *Gradient Boosting* e *XGBoost*. As variáveis listadas influenciam negativamente na estimativa do peso das estruturas, podendo reduzir o valor previsto.

Tabela 2 - Variáveis que reduziram a predição dos modelos de *Machine Learning*

Variável	***Random Forest***	***Gradient Boosting***	***XGBoost***
input17	-0.5465	-0.5465	-0.5465
input03	-	-0.2332	-0.2332
input18	-	-0.7095	-0.7095
input05	-	-	-0.509
input12	-	-	-0.474

Fonte: Elaboração própria, 2024.

Já a Tabela 3 apresenta as variáveis que aumentaram a predição dos modelos *Random Forest*, *Gradient Boosting* e *XGBoost*. As variáveis listadas influenciam positivamente na estimativa do peso das estruturas, podendo aumentar o valor previsto. A interpretação dessas contribuições é essencial para compreender o comportamento dos modelos e identificar padrões relevantes nos dados.

Tabela 3 - Variáveis que aumentaram a predição dos modelos de *Machine Learning*

Variável	***Random Forest***	***Gradient Boosting***	***XGBoost***
input02	0.8597	0.8597	0.8597
input12	-0.3829	-0.3829	-0.3829
input13	-0.5829	-0.5829	-0.5829
input10	-	-0.8188	-0.8188
input06	-	3.644	-
input16	-	-0.6781	-0.6781
input19	-0.3855	-0.3855	-0.3855
input20	-0.562	-0.562	-0.562

input09	-	-0.2863	-0.
input15	-	0.1096	-
input14	-0.5194	-	-
input00	-0.467	-	-

Fonte: Elaboração própria, 2024.

Ao analisar as Tabela 2 e 3, observa-se que algumas variáveis impactam de maneira consistente os três modelos, enquanto outras possuem influência distinta dependendo do algoritmo utilizado. A variável input17, por exemplo, reduz a predição em todos os modelos, com um impacto de -0.5465. Já a variável input02 aparece como um dos principais fatores que aumentam a predição em todas as abordagens, com um valor positivo de 0.8597.

O *Random Forest* apresenta um conjunto mais restrito de variáveis influentes em comparação com os demais modelos, sugerindo que sua capacidade de captura de padrões pode ser mais limitada ou distribuída entre um maior número de variáveis. O *Gradient Boosting*, por sua vez, apresenta um maior número de variáveis que aumentam e reduzem a predição, com a variável input06 destacando-se por um impacto positivo expressivo (3.644). O *XGBoost*, já que é um modelo derivado do *Gradient Boosting*, compartilha muitas das variáveis influentes, mas sem a presença de input06, o que pode indicar diferenças na forma como os modelos ponderam a importância das features.

Além disso, percebe-se que algumas variáveis possuem impactos negativos em um modelo, mas positivos em outro. Esse comportamento pode indicar que diferentes técnicas de aprendizado extraem relações distintas entre os atributos da base de dados e o peso estrutural previsto. Dessa forma, a análise conjunta dessas informações permite identificar quais variáveis são mais robustas na predição e quais podem estar sujeitas a variações dependendo do modelo empregado.

Com base nesses resultados, a escolha do modelo ideal deve levar em consideração não apenas a acurácia, mas também a interpretabilidade e o impacto das variáveis na predição.

4 CONSIDERAÇÕES FINAIS

A integração dos Algoritmos Genéticos com ajustes de modelos de *Machine Learning* tem o potencial de transformar as otimizações de estruturas metálicas, permitindo projetos mais precisos, econômicos e sustentáveis. Nos testes realizados, o *Gradient Boosting* e o *XGBoost* apresentaram desempenho superior ao Random Forest, com destaque para o *XGBoost*, que obteve o menor erro absoluto médio (*MAE*) de 4.7255 e explicação da variabilidade dos dados (R^2 *Score*) de 0.5809. O *Gradient Boosting* também demonstrou resultados robustos, com desempenhos semelhantes no *MSE* e *RMSE* e leve vantagem no R^2 *Score*.

Diante desses resultados, pode-se concluir que os modelos baseados em *boosting*, especialmente o *Gradient Boosting* e o *XGBoost*, são mais adequados para o problema em questão, pois apresentam menor erro e maior capacidade explicativa dos dados. No entanto, a escolha final do modelo deve considerar não apenas a performance, mas também fatores como tempo de treinamento e interpretabilidade, que podem variar conforme a aplicação desejada. Ainda pode-se argumentar que é importante nesse contexto fazer um estudo mais detalhado da capacidade preditiva dos modelos em um contexto *out-of-sample*, isto é, dividindo o banco de dados em conjuntos de treino e teste, e aferindo as métricas de predição no conjunto de teste (ou possivelmente em vários conjuntos de teste, fazendo-se o uso de técnicas de validação cruzada).

Para melhorar ainda mais a precisão dos modelos, futuras pesquisas podem explorar refinamentos na modelagem paramétrica, otimizando os hiperparâmetros dos modelos utilizados. Além disso, pode-se aprimorar o Algoritmo Genético para validar os modelos com base nas normas técnicas aplicáveis, como a norma brasileira para projeto de estruturas de aço e de estruturas mistas de aço e concreto de edificações (NBR 8080) e outras regulamentações relacionadas à construção metálica. Outras estratégias incluem o refinamento dos ajustes dos modelos com base no desempenho das variáveis mais influentes e na análise detalhada dos erros residuais, permitindo melhor compreensão do comportamento das predições.

Por fim, a incorporação de técnicas mais avançadas, como Redes Neurais Artificiais, pode ser uma alternativa promissora para aprimorar a acurácia das previsões e ampliar a aplicabilidade da abordagem desenvolvida neste trabalho. Assim, espera-se que este estudo

contribua significativamente para o avanço da otimização estrutural baseada em *Machine Learning*, proporcionando soluções mais eficientes e seguras para a engenharia de estruturas metálicas.

REFERÊNCIAS

BARBOSA, V. W. Modelagem paramétrica e análise estrutural de malha de madeira. 2018. Trabalho de Conclusão de Curso (Graduação em Engenharia Civil) - Universidade Federal de Campina Grande, Campina Grande, 2018. Disponível em: http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/23536. Acesso em: 15 mar. 2024.

CRUZ, P. J. S. Estruturas Metálicas: Conceitos, Técnicas e Aplicações. Rio de Janeiro: Elsevier, 2015.

DEB, K. Multi-Objective Optimization Using Evolutionary Algorithms. Chichester: John Wiley & Sons, 2001.

GOLDBERG, D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading: Addison-Wesley, 1989.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Deep Learning. Cambridge: MIT Press, 2016.

HOLLAND, J. H. Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press, 1975.

MITCHELL, M. An Introduction to Genetic Algorithms. Cambridge: MIT Press, 1998.

THAI, H.-T. Machine learning for structural engineering: A state-of-the-art review. Structures, Amsterdam, v. 38, p. 448-491, abr. 2022. Disponível em: https://doi.org/10.1016/j.istruc.2022.02.003. Acesso em: 22 maio 2024.

.

APÊNDICE A - Código do Algoritmo Genético

```
import scriptcontext as sc
import Rhino as rc
import os
import System

from random import random, randint


class Parametro():

    def __init__(self, nome, minimo, maximo):  # o valor entra nesse
algoritmo
        self.nome = nome  # parametro nome. Ex.: largura
        self.minimo = minimo  # parametro min. Ex.: 1
        self.maximo = maximo  # parametro max. Ex.: 10


class Individuo():
    def __init__(self, minimos, maximos, geracao=0):
        self.minimos = minimos
        self.maximos = maximos
        self.nota_avaliacao = 0
        self.geracao = geracao
        self.cromossomo = []

        for i in range(len(lista_parametros)):
            if (type(minimos[i]) == float) and (type(maximos[i]) == float):
                self.cromossomo.append(random.uniform(
                    minimos[i], maximos[i]))  # instance
            else:
                self.cromossomo.append(randint(minimos[i], maximos[i]))

    def avaliacao(self):

        nota = randint(1000, 2222)
        self.nota_avaliacao = nota

    def crossover(self, outro_individuo):

        corte = int(round(random() * len(self.cromossomo)))

        filho1 = outro_individuo.cromossomo[0:corte] +
self.cromossomo[corte:]
        filho2 = self.cromossomo[0:corte] +
```

```
outro_individuo.cromossomo[corte::]
        filhos = [Individuo(self.minimos, self.maximos, self.geracao + 1),
                  Individuo(self.minimos, self.maximos, self.geracao + 1)]

        filhos[0].cromossomo = filho1
        filhos[1].cromossomo = filho2

        return filhos

    def mutacao(self, taxa_mutacao, minimos, maximos):
        for i in range(len(self.cromossomo)):
            if random() < taxa_mutacao:
                if (type(minimos[i]) == float) and (type(maximos[i]) ==
float):
                    self.cromossomo[i] = random.uniform(
                        minimos[i], maximos[i])  # instance
                else:
                    self.cromossomo[i] = randint(minimos[i], maximos[i])
        return self

class AlgoritmoGenetico():

    def __init__(self, tamanho_populacao, objetivo):
        self.tamanho_populacao = tamanho_populacao
        self.objetivo = objetivo
        self.populacao = []
        self.geracao = 0
        self.melhor_solucao = 0

    def inicializa_populacao(self, minimos, maximos):
        for i in range(self.tamanho_populacao):
            self.populacao.append(Individuo(minimos, maximos))
        self.melhor_solucao = self.populacao[0]

    def ordena_populacao(self):
        if self.objetivo == 'minimizar':
            reverse = True
        else:
            reverse = False
        self.populacao = sorted(self.populacao,
                                key=lambda populacao:
populacao.nota_avaliacao,
                                reverse=reverse)

    def melhor_individuo(self, individuo):
```

```
        if self.objetivo == 'maximizar':
            if individuo.nota_avaliacao >
self.melhor_solucao.nota_avaliacao:
                self.melhor_solucao = individuo
        else:
            if individuo.nota_avaliacao <
self.melhor_solucao.nota_avaliacao:
                self.melhor_solucao = individuo

    def soma_avaliacoes(self):
        soma = 0
        for individuo in self.populacao:
            soma += individuo.nota_avaliacao

        return soma

    def seleciona_pai(self, soma_avaliacao):
        pai = -1
        valor_sorteado = random() * soma_avaliacao
        soma = 0
        i = 0
        while i < len(self.populacao) and soma < valor_sorteado:
            soma += self.populacao[i].nota_avaliacao
            pai += 1
            i += 1

        return pai

    def visualiza_geracao(self):

        for i in range(ag.tamanho_populacao):
            cromossomo = str(self.populacao[i].cromossomo)
            nota = self.populacao[i].nota_avaliacao

            id = self.populacao[0].geracao + i

            print(f"\n******* Individuo {id} *******")
            print(f"Cromossomo: {cromossomo}")
            print(f"Nota: {nota}\n")

        melhor = self.populacao[0]
        print(
            f"\nG: {self.populacao[0].geracao} | Cromossomo:
{melhor.cromossomo}")

        return cromossomo, nota
```

```
    def resolver(self, taxa_mutacao, numero_geracoes, minimo, maximo, run):
        if run == False:

            self.inicializa_populacao(minimo, maximo)
            for individuo in self.populacao:
                individuo.avaliacao()

            self.ordena_populacao()

            self.visualiza_geracao()

        else:

            self.inicializa_populacao(minimo, maximo)
            for individuo in self.populacao:
                individuo.avaliacao()

            self.ordena_populacao()
            self.visualiza_geracao()

            for geracao in range(numero_geracoes):
                soma_avaliacao = self.soma_avaliacoes()
                nova_populacao = []

                for individuos_gerados in range(0, self.tamanho_populacao,
2):
                    pai1 = self.seleciona_pai(soma_avaliacao)
                    pai2 = self.seleciona_pai(soma_avaliacao)

                    filhos = self.populacao[pai1].crossover(
                        self.populacao[pai2])

                    nova_populacao.append(filhos[0].mutacao(
                        taxa_mutacao, minimo, maximo))
                    nova_populacao.append(filhos[1].mutacao(
                        taxa_mutacao, minimo, maximo))

                self.populacao = list(nova_populacao)

                for individuo in self.populacao:
                    individuo.avaliacao()

                self.ordena_populacao()
                self.visualiza_geracao()
                melhor = self.populacao[0]
```

```
                self.melhor_individuo(melhor)

            print(f"\nMelhor solução: %s" % self.melhor_solucao.cromossomo,
                  "Melhor nota: %s" % self.melhor_solucao.nota_avaliacao)

        return self.melhor_solucao.cromossomo


if __name__ == '__main__':
    lista_parametros = []
    lista_parametros.append(Parametro('Input00', 2, 10))
    lista_parametros.append(Parametro('Input01', 5, 10))
    lista_parametros.append(Parametro('Input02', 2, 20))
    lista_parametros.append(Parametro('...', 10, 1000))
    lista_parametros.append(Parametro('InputN', 2, 5000))
    nome = []
    minimo = []
    maximo = []

    for parametro in lista_parametros:
        nome.append(parametro.nome)
        minimo.append(parametro.minimo)
        maximo.append(parametro.maximo)

    tamanho_populacao = 59
    objetivo = 'minimizar'
    taxa_mutacao = 0.05
    numero_geracoes = 100

    ag = AlgoritmoGenetico(tamanho_populacao, objetivo)

    run = True

    resultado = ag.resolver(taxa_mutacao, numero_geracoes, minimo, maximo,
run)
```

APÊNDICE B - Código de Raspagem de Dados

```
import scriptcontext as sc
import Rhino as rc
import os
import System

def checkOrMakeFolder():
    if ghdoc.Path:
        folder = os.path.dirname(ghdoc.Path)
        ghDef = ghenv.LocalScope.ghdoc.Name.strip("*")
        captureFolder = folder + "\\" + str(ghDef)
        if not os.path.isdir(captureFolder):
            os.makedirs(captureFolder)
        return captureFolder

def makeFileName():
    return str(goalValue)

def captureActiveViewToFile(width,height,path):
    sc.doc = rc.RhinoDoc.ActiveDoc
    activeView = sc.doc.Views.ActiveView
    imageDim = System.Drawing.Size(width,height)
    try:
        imageCap =
rc.Display.RhinoView.CaptureToBitmap(activeView,imageDim)
        System.Drawing.Bitmap.Save(imageCap,path)
        rc.RhinoApp.WriteLine(path)
        return path
    except:
        raise Exception(" Capture failed, check the path")

if Toggle:
    capFolder = checkOrMakeFolder()
    fileText = os.path.join(capFolder + ".csv")
    with open(fileText, 'a') as file_object:
        fileName = makeFileName()
        count = 0
        try:
            path = os.path.join(capFolder,fileName + ".png")
            Path = captureActiveViewToFile(width,height,path)
        except:
            raise Exception("Capture failed, save the GH definition")
        converte = ', '.join(map(str, data))
        concatenatedData = converte + ", " + fileName + "\n"
        file_object.write(concatenatedData)
```

APÊNDICE C - Código de Análise Exploratória

```
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Create the heatmap
plt.figure(figsize = (10,8))
sns.heatmap(correlation_matrix, cmap = 'coolwarm')
plt.show()

# Criar subplots organizados
fig, axes = plt.subplots(7, 3, figsize=(12, 17))  # Grid de 5x5 para 22 variáveis
axes = axes.flatten()

# Gerar histogramas para as 22 variáveis explicativas
for i, col in enumerate(df.columns[:-1]):  # Excluindo a variável resposta
    sns.histplot(df[col], bins=30, kde=True, ax=axes[i], color="teal")
    axes[i].set_title(col, fontsize=10)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("")

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

APÊNDICE D - Código de Preparação dos Dados

```
import pandas as pd
from google.colab import files
from IPython.display import Image
import shap


df = pd.read_csv('resultados_tratados.csv', usecols=lambda column: column !=
'Unnamed: 0')

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


X = df.drop('goal', axis=1)
y = df['goal']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Standardize the data (important for most machine learning models)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert back to DataFrame for better interpretability
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
print(X_train_scaled.head())
```

APÊNDICE E - Código do modelo *Gradient Boosting*

```
from sklearn.ensemble import GradientBoostingRegressor

# Treinar um modelo Gradient Boosting
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
max_depth=5, random_state=42)
gb_model.fit(X_train_scaled, y_train)

# Fazer previsões
y_pred_gb = gb_model.predict(X_test_scaled)

# Avaliar o modelo
mae_gb = mean_absolute_error(y_test, y_pred_gb)
mse_gb = mean_squared_error(y_test, y_pred_gb)
rmse_gb = np.sqrt(mse_gb)
r2_gb = r2_score(y_test, y_pred_gb)

# Exibir as métricas
print(f'Gradient Boosting - MAE: {mae_gb:.4f}')
print(f'Gradient Boosting - MSE: {mse_gb:.4f}')
print(f'Gradient Boosting - RMSE: {rmse_gb:.4f}')
print(f'Gradient Boosting - R² Score: {r2_gb:.4f}')

# Initialize SHAP explainer for XGBoost
explainer = shap.Explainer(gb_model, X_train_scaled)
# Calculate SHAP values for the test set
shap_values = explainer(X_test_scaled)
# Generate a summary plot
shap.summary_plot(shap_values, X_test_scaled)

# Visualize SHAP values for a single prediction
shap.initjs()  # Required for interactive visualization in notebooks

# Choose an index for the test instance
index = 0
# Generate a force plot for the first test instance
shap.force_plot(explainer.expected_value, shap_values[index].values,
X_test_scaled.iloc[index])

# Initialize the LIME explainer
explainer_lime = lime.lime_tabular.LimeTabularExplainer(
    X_train_scaled.values,
    feature_names=X_train.columns,
    class_names=['MedHouseVal'],
    verbose=True,
    mode='regression'
)
```

```
# Choose an instance to explain
i = 0
# Generate LIME explanation for the i-th test instance
exp = explainer_lime.explain_instance(X_test_scaled.iloc[i].values,
gb_model.predict)
# Show the LIME explanation in text form
exp.show_in_notebook(show_table=True)
```

APÊNDICE F - Código do modelo *XGBoost*

```
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Treinar um modelo XGBoost
model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5,
random_state=42)
model.fit(X_train_scaled, y_train)

# Fazer previsões
y_pred = model.predict(X_test_scaled)

# Avaliar o modelo
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # RMSE é a raiz quadrada do MSE
r2 = r2_score(y_test, y_pred)

# Exibir as métricas
print(f'Mean Absolute Error (MAE): {mae:.4f}')
print(f'Mean Squared Error (MSE): {mse:.4f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.4f}')
print(f'R² Score: {r2:.4f}')


# Initialize SHAP explainer for XGBoost
explainer = shap.Explainer(model, X_train_scaled)
# Calculate SHAP values for the test set
shap_values = explainer(X_test_scaled)
# Generate a summary plot
shap.summary_plot(shap_values, X_test_scaled)

# Visualize SHAP values for a single prediction
shap.initjs()  # Required for interactive visualization in notebooks

# Choose an index for the test instance
index = 0
# Generate a force plot for the first test instance
shap.force_plot(explainer.expected_value, shap_values[index].values,
X_test_scaled.iloc[index])

import lime
import lime.lime_tabular

# Initialize the LIME explainer
explainer_lime = lime.lime_tabular.LimeTabularExplainer(
```

```
    X_train_scaled.values,
    feature_names=X_train.columns,
    class_names=['MedHouseVal'],
    verbose=True,
    mode='regression'
)
# Choose an instance to explain
i = 0
# Generate LIME explanation for the i-th test instance
exp = explainer_lime.explain_instance(X_test_scaled.iloc[i].values,
model.predict)
# Show the LIME explanation in text form
exp.show_in_notebook(show_table=True)
```

APÊNDICE G - Código do modelo *Random Forest*

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Treinar um modelo Random Forest
rf_model = RandomForestRegressor(n_estimators=100, max_depth=5,
random_state=42)
rf_model.fit(X_train_scaled, y_train)

# Fazer previsões
y_pred_rf = rf_model.predict(X_test_scaled)

# Avaliar o modelo
mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, y_pred_rf)

# Exibir as métricas
print(f'Random Forest - MAE: {mae_rf:.4f}')
print(f'Random Forest - MSE: {mse_rf:.4f}')
print(f'Random Forest - RMSE: {rmse_rf:.4f}')
print(f'Random Forest - R² Score: {r2_rf:.4f}')


# Initialize SHAP explainer for XGBoost
explainer = shap.Explainer(rf_model, X_train_scaled)
# Calculate SHAP values for the test set
shap_values = explainer(X_test_scaled)
# Generate a summary plot
shap.summary_plot(shap_values, X_test_scaled)

# Visualize SHAP values for a single prediction
shap.initjs()  # Required for interactive visualization in notebooks

# Choose an index for the test instance
index = 0
# Generate a force plot for the first test instance
shap.force_plot(explainer.expected_value, shap_values[index].values,
X_test_scaled.iloc[index])

# Initialize the LIME explainer
explainer_lime = lime.lime_tabular.LimeTabularExplainer(
    X_train_scaled.values,
    feature_names=X_train.columns,
    class_names=['MedHouseVal'],
```

```
    verbose=True,
    mode='regression'
)
# Choose an instance to explain
i = 0
# Generate LIME explanation for the i-th test instance
exp = explainer_lime.explain_instance(X_test_scaled.iloc[i].values,
rf_model.predict)
# Show the LIME explanation in text form
exp.show_in_notebook(show_table=True)
```